Complete Programs Included

NO. 75

U.S. Edition: $2.50
International Edition: $3.50

SEPTEMBER 1984

MICRO™

for the Serious Computerist

Structure Tree Utility
Introduction to FORTH
FORTH Multi-Tasking
Time Series Forecasting
TWERP

0 74470 11690 0

NOW SHOWING

1541 FLASH!

AT COMMODORE 64™ DEALERS

FLASH! Gets the Gold at the Computer Olympics

The Skyles Electric Works *1541 FLASH!* dashed off with the gold at the Computer Olympics here.

The *1541 FLASH!* loaded and saved programs and files *three times faster* than an unenhanced Commodore 1541 disk drive could. Faster than any other disk drive with compatible disk format.

"...faster than any other disk drive..."

Three times faster! The device delighted the home crowd, which watched the *1541 FLASH!* set a meet record, and leave its competition in the dust.

Once installed, the *1541 FLASH!* is transparent. Computer operations all remain unaffected as it speeds up every disk-related function. The *FLASH!* is a permanent installation with both a software (ROM) and a hardware component. Through keyboard commands or a hardware switch, you can even return to the old, slow loading method—if you really want to.

And there is nothing new to learn for the *FLASH!* No special tricks or techniques. Once it's in, just watch it go.

But if you're really serious about programming, the *1541 FLASH!* is a gold mine. The manual will show you how to write software allowing data transfer to and from the 1541 disk drive at speeds up to *10 times* the normal.

For programs that usually load with a " "*",8,1" command, just hit Shift/Run-Stop. A spreadsheet program like BUSICALC 3 then loads in about 25 seconds.

The *1541 FLASH!* even adds 21 extra commands for the Commodore 64 user. Some of these include editing, programming and loading commands, as well as "DOS Wedge" commands. You can ignore all these commands, though, and just enjoy the rapid disk operations.

It wowed the crowd at the Computer Olympics. Once you see its sheer speed, you'll know why. Call its coach, Skyles Electric Works, to place your order or to get more info.

***1541 FLASH!*, an add-on assembly, for the Commodore 64/1541 costs only $79.95.**

Skyles Electric Works
231E South Whisman Road
Mountain View, CA 94041
(415) 965-1735

Available from your local Commodore 64 dealer or call 1-800-227-9998.

1541 FLASH! is a trademark of Skyles Electric Works.
Commodore 64 is a trademark of Commodore.

Want to become an Apple expert?

Join *the* club.

A.P.P.L.E.

Apple PugetSound Program Library Exchange

The Apple PugetSound Program Library Exchange is the world's first, oldest, and largest Apple computer user group. Our membership is comprised of Apple enthusiasts throughout the world, and we provide support for all levels of technical ability, from beginner to seasoned program author.

A membership in A.P.P.L.E. will bring the Apple owner 7 day per week hotline privileges for technical assistance **when you need it,** plus the international magazine Call—A.P.P.L.E., and incredible discounts on our fully supported, low priced, world famous software products, and hardware.

A.P.P.L.E. is a member owned, non-profit service organization. Write today for a free copy of our magazine and club information, or join by filling out the enrollment coupon

VALUESOFT

A product whose time has come.

Fully supported, guaranteed, high quality software at low prices.

VALUESOFT INCLUDES:

Uncommon at

$12.50 per disk.

program contains its own document on disk.

2 word processing programs
exciting games
useful utilities
high quality graphics
finance and education

The quality VALUESOFT line of software is available through the Apple PugetSound Program Library Exchange.

NEW *IIc* OWNER?

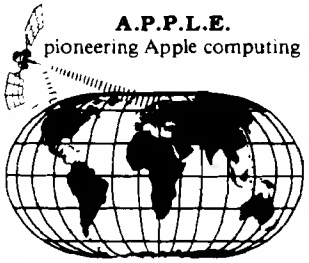
programs for the *IIc* now in stock

Graphics
Word Processing
Utilities
Database

Join Now and Receive 10 FREE (5-1/4")

Apple II, II+, *IIe, IIc, III,* Lisa and Macintosh are all registered trademarks of Apple Computer Inc.

A.P.P.L.E.
pioneering Apple computing



since 1978.

Mail to
A.P.P.L.E
21246 - 68th Ave. S.
Kent, WA 98032
(206) 872-2245
or call our toll-free number
1-800-426-3667
(24 Hrs. Orders Only)

☐ MEMBERSHIP $26 one-time application fee + $25 first year dues. $51

☐ FREE INFO + Call—A.P.P.L.E
Please send free information

Name _____

Address _____

City _____

State _____ Zip _____

Phone # _____

M/C VISA # _____

Exp. Date _____

Additional foreign postage required for membership outside the U.S.

highlights

Featured This Month

This month we enter into the world of FORTH, in addition to a varied selection of other exciting subjects, programs and projects.

Time-Series Forecasting — enter into the world of Nostradamus with the sagacity of Einstein. This program uses various forecasting techniques to predict the future, from interest rates to the Dow Jones. Includes versions for the Apple, Atari, Commodore 64 and CoCo.

Introduction to FORTH — a look at the world of FORTH. Who would be interested in FORTH, why, what features does it offer, its uses and implementations.

Textfile Write Edit Read Program — for all of you Apple users who have cursed when they wanted to read a text file and only got an error message. This program is an invaluable help in writing, reading, and editing text files.

Multi-Tasking for FORTH — this program allows seperate tasks to run in the 'background' while still having the FORTH interpreter available in the 'foreground.' A concept and program with powerful implications.

Transforming dBase II Files — find out how to alter your dBase II files so that you can use them with Wordstar/Mailmerge to produce personalized letter forms.

Stepper — who has step-traced his way through an assembler program only to have to suffer through jumps to monitor routines? This program saves from this time consuming annoyance, making debugging a little more enjoyable.

Structure Tree Utility — nothing is worse than forgetting the calling structure of a word in FORTH. With this program you can easily recall which came first — the chicken or the egg.

68000 Exception Processing — how to take advantage of the 68000's capabilities, software exception processing in general and hardware exception processing for the SAGE.

Graphic Print for C64 Part 3 — the final installment of this excellent series provides the programs and techniques required to generate full color pictures on your standard printer!

Approximating the Square Root of the Sum of Squares — a fast and time saving method for dealing with that candidate for 'the most often used calculation' — the square root.

MICRO

CONSUMERS GUIDE 'ONE OF THE BEST PROGRAMS AVAILABLE FOR THE APPLE COMPUTERS'
SOFTALK 'IT PAYS FOR ITSELF'
APPLE ORCHARD 'FIRST RATE INTERNAL AND EXTERNAL MAINTENANCE OF YOUR COMPUTER'
POPULAR COMPUTING 'FIRST LEVEL DIAGNOSTICS PACKAGE'
NIBBLE MAGAZINE 'THIS PACKAGE SHOULD BE IN THE LIBRARY OF EVERY APPLE USER'
MATHEMATICS & COMPUTER EDUCATION 'EASY TO USE, A PERFECT PACKAGE'
IN CIDER 'DOES ONE JOB VERY WELL INDEED'

MASTER DIAGNOSTICS

There is only one thing more important than your apple®

Maintaining it.

HOW MANY DISKETTES HAVE YOU INITIALIZED WITH YOUR DISK DRIVES RUNNING TOO FAST OR TOO SLOW? THINK ABOUT WHAT THAT COULD MEAN

DID YOU KNOW THAT THE DRIVE SPEED OF YOUR APPLE SHOULD BE AS CLOSE TO 300 RPM AS POSSIBLE? LIKE A RECORD OR TAPE SYSTEM VARIES WITH MOTOR SPEED SO DOES A DISKETTE

WHEN WAS THE LAST TIME YOU CLEANED THE READ/WRITE HEADS OF YOUR DRIVES? THEY SHOULD HAVE BEEN CLEANED LAST MONTH AND WITH OUR PROGRAMMED UTILITIES YOU COULD DO SO AT THE PUSH OF A BUTTON

HOW ABOUT THE WRITE PROTECT SWITCH? IS IT WORKING PROPERLY SO YOU WON'T DESTROY YOUR PROGRAM DISKETTE OR PROTECTED DATA?

THERE'S LOTS MORE AND IT WILL ONLY TAKE 15 MINUTES A MONTH TO KEEP YOUR HIGH TECHNOLOGY EQUIPMENT RUNNING AT HIGH PERFORMANCE. PREVENT PROBLEMS OR DIAGNOSE PROBLEMS AND SAVE YOURSELF ONE OF THOSE DAYS

WITH MASTER DIAGNOSTICS ANYONE CAN DO IT.

THE PROGRAM THAT PAYS FOR ITSELF

WHEN ORDERING SPECIFY
version II & II plus or version //e

MasterCard VISA

☐ master diagnostics $55.00

☐ master diagnostics + plus $75.00

DIAL 1-800-835-2246

THE TESTS INCLUDE

MOTHERBOARD ROM TEST
APPLESOFT CARD TEST*
INTEGER CARD TEST
MOTHERBOARD RAM TEST
16K RAM CARD TEST*
AUX RAM TEST*
80 COLUMN CARD TEST*
PARALLEL CARD TEST*
SPEAKER FUNCTION TEST
SQUARE WAVE MODULATION
ON BOARD HELP

DISK DRIVE SPEED CALIBRATION
DRIVE HEAD READ/WRITE TEST
WRITE PROTECT SWITCH TEST
DRIVE HEAD CLEANING ROUTINES
DISK DRIVE MAINTENANCE
DC HAYES MICROMODEM II TEST
PADDLE & SPEAKER TEST
PADDLE & BUTTON TEST
PADDLE DRIFT TEST
INTERNAL MAINTENANCE
FORTY PAGE MANUAL

MONITOR SKEWING TESTS
MONITOR & MODULATOR CALIBRATION
MONITOR TEXT PAGE TEST
MONITOR TEST PATTERN
MONITOR & TV YOKE ALIGNMENT
LO RES COLOR TESTS
HI RES COLOR TESTS
LISSAJOUS PATTERNS
FIND HI RES COLOR GENERATOR
GENERAL MAINTENANCE
*APPLE //E

THE + PLUS

Master Diagnostics + Plus provides everything needed to maintain your computer. The entire package is housed in our own molded case to protect against static electricity, x-ray and other contaminants.

Included in the kit is:
- THE DIAGNOSTICS DISKETTE
- FORTY PAGE PROCEDURE MANUAL
- HEAD CLEANING KIT
- CRT SCREEN CLEANER
- COMPUTER/DRIVE HOUSING CLEANER
- REUSEABLE CHAMOIS TIPPED WANDS

NIKROM®
Technical Products, Inc.
25 Prospect Street, Leominster MA 01453

FREE OFFER

Buy this 68000 computer, and we'll give you a 65C02...free.

The MTU-140 has always been a good deal, because it's such a versatile computer. Now when you purchase an MTU-140 ($3995) you get the new 65C02 processor chip free. And, by adding the DATA-MOVER board ($1080) you'll have a 68000 computer to boot . ...plus a total of 335KB RAM to share between them. The 65C02 is just like the 6502 plus 8 new instructions and 2 new addressing modes using the latest high speed CMOS technology. With the CODOS/DMXMON operating system, you choose which processor is in control...or have both running simultaneously, each performing flawlessly...the way MTU computers have always worked.

Our customers are sold on the MTU-140. They know we designed it for thinking people ...to extend their abilities. They have found it easy to customize for their own needs ...providing solutions they had not thought possible. We think you will agree...the MTU-140 is tailor-made for people who need their own professional computer.

The MTU-140 is fast. At the touch of a button, you can load 32KB in just 2.6 seconds...and another 20KB in just 1 second! Plus, both CPU's can perform other tasks during disk operations . . . even service interrupts. Few computers offer this freedom...none with both 65C02 and 68000 processors! **Equipped with the DATAMOVER, the MTU-140 provides:**

- 8MHz 68000 plus 256KB RAM with 2 DMA ports
- 1MHz 65C02 plus 80KB RAM with multiple DMA ports
- Dual 1MB 8" Floppies for ultra-reliable operation
- DMA disk operations into memory, no CPU interference
- DMA hi-res Graphics/Text display, no CPU interference
- 96 key detached keyboard with 5 separate cursor keys
- 8-bit D/A speech and music port
- Fiber optic light pen
- 2 Parallel, 1 serial port
- Internal expansion card slots
- CODOS/DMXMON Operating system using device independent channels
- Full screen, bidirectional scroll editor, handles 1MB files
- MTU-BASIC with graphic & disk extensions
- Communications with attended/unattended use—source available
- Graphic editor, slideshow presentation
- 4-part harmony instrument synthesizer music
- Over 50 machine language utilities

Choose from our extensive selection of software...for the 65C02: MTU-BASIC (std), MTU-C ($200), MTU-FORTH ($79), 6502/65C02/6511 macro assembler ($150),...for the 68000: MTU-BASIC 1.5 ($50), Motorola compatable, macro cross-assembler ($200), FORTH68K-83 (with source $250), FORTH68K-83 META COMPILER ($250),...68000 MAGIC/L ($495), Digital Research CP/M68K with C language ($call). For word processing, WORDPIC ($420) mixes graphics with text. And you can run any CP/M 2.2 program with the PROGRAM-MOVER Board (MTU-CP/M, Z-80A, 64KB RAM-$650).

For laboratory use, the MULTI-O System ($1500) offers IEEE-488 bus, clock calendar, 2 parallel & 2 serial ports, 12 bit D/A & A/D (8 channels) with up to 16,000 samples/second, 5,000/second sustained to disk from MTU-BASIC! It is fully enclosed... including connectors, power supply and all driver software for interfacing all devices to CODOS and all languages. And for signal analysis work, our DigiSound-16 offers 16-bit linear, 100,000 samples/second, variable internal or external sample clock, 64KB DMA RAM buffer, and parallel port interface to work with any computer ($2995).

Interested? Call or write MTU today for details. We just made a good deal even better...with a free 65C02.

MTU

Micro Technology Unlimited
2806 Hillsborough St.
Raleigh, North Carolina 27607
(919) 833-1458

Publisher/Editor-In-Chief
Robert M. Tripp

Associate Publisher
Cindy Kocher

Production Manager
Jennifer Collins

Technical Editor
Mark S. Morano

Technical Editor
Mike Rowe

Advertising Manager
William G. York

Circulation Manager
Linda Hensdill

Office Manager
Pauline Giard

Shipping Director
Marie Ann Wessinger

Comptroller
Donna M. Tripp

Accounting
Louise Ryan

Contributing Editors
Cornelis Bongers
Phil Daley
David Malmberg
John Steiner
Jim Strasma
Paul Swanson
Richard C. Vile, Jr.
Loren Wright

Dealer Sales Representative
Alison Churchill

MICRO is published monthly by:
MICRO, Chelmsford, MA 01824.
Second Class postage paid at:
Chelmsford, MA 01824 and additional mailing offices.
USPS Publication Number: 483470.
ISSN. 0271-9002.
Send subscriptions, change of address, USPS Form 3579, requests for back issues and all othe fulfillment questions to:
MICRO
P.O.Box 6502
Chelmsford, MA 01824
or call 617/256-3649.
Subscription Rates: (per year):
U.S. $24.00 or $42.00 for two years
Foreign surface mail: $27.00
Air mail: Europe $42.00
Mexico, Central America, Middle East, North Africa, Central Africa $48.00
South America, South Africa, Far East, Australia, New Zealand $72.00

Copyright © 1984 by MICRO.
All Rights Reserved.

MICRO

SEPTEMBER

15 Introduction to FORTH
Kenneth Butterfield
The basic Why's and Wherefore's about the FORTH language.

18 Multi-Tasking in FORTH
Kenneth Butterfield
A Technique and Program for Running Multiple Tasks Under FORTH.

24 Structure Trees in FORTH
Michael Dougherty
A FORTH Utility that Prints the Structure of a FORTH Word.

27 Textfile Write Edit Read Program (T.W.E.R.P)
N. D. Greene
Now reading, writing, and editing textfiles is easy.

31 Graphic Print for C-64, Part 3
Michael J. Keryan
Add Full Color to Your Graphic Printouts -Without a Color Printer.

36 Approximating the Square Root of the Sum of the Squares
Chris Williams
A Very Fast Method of Calculating this Useful Function.

44 Interface Clinic: A Major Hardware Interface
Ralph Tenny
Design a major hardware interface - a receiver board for the 32K CoCo.

NO. 75

47 **68000 Exception Processing**
Mike Rosing
How 68000 Uses Exception Processing to Handle Software and Hardware.

52 **Transferring dBase II Files For Use With Wordstar/Mailmerge**
Robert R. Carroll
Alter your dBase II files and use them to produce personalized letter forms.

55 **Stepper**
Chester H. Page
Step-Trace facility that allows you to bypass monitor routines.

61 **Time-Series Forecasting**
Brian Flynn
A Program to Predict the Future - That Runs on the Apple, Atari, C64 or CoCo.

Product Reviews

13 Wizard — A game utilizing the C64's capabilities to the maximum — that even lets the player create his own material.

13 Person-to-Person — A feature packed communication package for the Apple, that includes a mailing list capability.

13 Mail Now! — A user-friendly mailing list program for the C64.

14 Stocker1 — For the serious stock market investor, a program for predicting performance that can interact with a time-shared data base.

14 MMP-1000C Modem — A full-feature communications package for any Atari, including all required hardware and software.

Departments

2 Highlights
7 Guest Editorial
8 Editorial
10 Feedback
11 Spotlight
68 Books
69 Catalog
71 Lyte Bytes
72 Advertiser Index
72 Coming in October

From the editors of
A.N.A.L.O.G. Computing

$14.95

THE ANALOG COMPENDIUM

The best ATARI® Home Computer Programs from the first ten issues of A.N.A.L.O.G. Computing Magazine.

All Compendium programs are available on DISK.

The **ANALOG Compendium** is available at selected book and computer stores, or you can order it direct. Send a check or money order for $14.95 + $2 shipping and handling to: **ANALOG Compendium, P.O. Box 615, Holmes, PA 19043.**

Or you can order by phone with MasterCard or VISA. Call toll free: 1-800-345-8112 (in PA, call 1-800-662-2444). **For orders outside the U.S., add an additional $5 air mail, $2 surface.**

why FORTH

by Mike Dougherty
Littleton, Colorado

Languages like FORTH represent an important turning point for computer users. Until now, computer usage was managed as a "closed shop." Users desiring computer aid were faced with either working with a software expert who might not understand their problems, or becoming their own software expert at the expense of effort applied to the original problem. In either case, the actual solution to the user problem was hindered.

In the last decade, the closed shop cycle was being discarded by several new software environments. Working in a scientific field which changed far faster than conventional software support, Charles Moore developed the language and operating system FORTH. In a sense, this language placed software development in the hands of the user. Users were able to quickly develop and modify software without the software middle man. Parallel to FORTH, the software engineer was also finding similar help in the environment of UNIX and C. The UNIX/C combination allowed the nebulous field of software support to be directly used by the software engineer. UNIX/C allowed each software engineer to build the tools needed to analyze, design, write, test and document software effectively. In education, Seymor Papert was developing an environment called LOGO to teach concepts in mathematics and programming. Papert's methods were based upon the students writing their own software rather than being passive subjects for CAI drill and practice.

From the viewpoint of the user, these languages represent the true second generation of software — languages which allow the user to directly apply a computer to user problems. These second generation languages may be characterized by the "toolbox" approach to problem solving. Each of the new computer environments allows the user to construct individual tools (functions, modules, etc.) which may be combined to solve problems. Instead of relying on an intermediate software engineer to design and build a single program, the second generation languages allow the user to build upon past software to solve new problems.

The toolbox approach becomes a software metaphor for the normal human process of learning. Learning typically consists of building upon previous knowledge. For example, speech must be learned in steps, each step building upon the last. A small child will first learn simple nouns and verbs. These words will be combined into short sentences to express desires. Adjectives, adverbs and prepositional phrases are later added to express more complex desires or ideas. Thus, learning to speak is a process of building new speech tools upon those previously learned. This process of synthesis is a natural and well practiced process for most people. The toolbox approach utilizes this concept for software development. (It should be pointed out that this "bottom up" methodology is not perfect. Just as many children who can talk must still be taught "proper" English in school, the building blocks must be rearranged or modified when the goal at the top is missed.)

Notice that the toolbox approach complements the normally advocated "top down" methodology. This is not bad. Rather, it is a reflection of two different problem environments. Top down programming was developed to aid software engineers dealing with complex programming tasks. Complex software is not limited to large government projects — even the Apple Lisa's integrated software required a 200 man-year effort! These systems require a different methodology than used in a laboratory or learning environment.

Given a set of requirements, a complex project may be successfully decomposed, layer by layer, into small, easily programmed and tested units. This decomposition will detail the interfaces between software functions and allow different portions of the software to be developed by different programmers or teams of programmers.

Although validated by several pilot projects and used in many actual projects, this top down methodology (called structured analysis and structured programming by Yourdon) requires two strong foundations. First, the software requirements must behave as a "damped oscillator." Although many requirements change throughout a project, these changes must converge to zero. That is, for a final product to be generated, the final requirements must be set prior to delivery. (In the real world, this is not always the case!) Top down methods are not very effective when requirements are vague.

Secondly, the right personnel must be found to perform the software decomposition. Proper decomposition requires a well structured, highly analytical thought process. Unfortunately, most of us do not adequately possess the skill and talent required for this job. In my limited experience with complex projects, the front end decomposition is, simply stated, very difficult. Not only is the decomposition difficult to do, there are problems in determining whether the decomposition is correct or even complete. Finally, the ultimate project success, years down the schedule, will depend upon proper and valid decomposition. Good software analysts can easily be worth their weight in gold.

Who are the readers of MICRO? I suspect that most of the readers fall into the toolbox category. Like myself, they are interested in using their personal computers for solving the wide range of problems encountered in everyday living, not programming complex U.S. Defense Projects. Since everyday problems cannot always be anticipated, the software requirements change continuously. In addition, it is difficult and cost prohibitive to find a software "middle man" with as much knowledge as the user. Only the user fully knows the problems to be solved. The toolbox approach simply makes good sense for most MICRO readers.

Where does this leave MICRO? I think that MICRO should expand to cover these new software environments. Versions of FORTH are available on most micros; similar versions of LOGO are supported on both Atari and Apple (LOGO Computer Systems, Inc.); UNIX is rapidly becoming the de facto operating system for the 68000

microprocessor family. Details of specific implementations as well as the general philosophy behind each software environment need to be covered. Each language has its own niche with unique advantages and disadvantages. The languages discussed here are not free — they do require investment in software, hardware and effort. By covering these second generation user languages, MICRO will allow the readers to determine what they should invest in their current or future personal computers. These languages are receiving attention from many sources and I feel that MICRO must seriously consider their coverage.

Mike Dougherty has worked in the software field since 1977 and is currently a Software Engineer with Martin Marietta Denver Aerospace in Colorado. He has specialized in developing real time data acquisition and control systems. While most of his software has been written in assembly language, in the last few yers he has used FORTH for work and personal projects. He has submitted several FORTH programs/articles, the first of which, *Structure Trees* appears in this issue.

[Editor's Note: This 'Guest Editorial' is essentially a very thoughtful letter MICRO received from the author in April. Since it expressed many of our thoughts very eloquently, we obtained permission to use it as an editorial.]

Discover Forth

Join the **FORTH Interest Group**

The FORTH Interest Group (FIG) is a non-profit member-supported organization, devoted to the Forth computer language. Join our 4700+ members and discover Forth. We provide our members with the information and services they need, including:

Over fifty local FIG chapters (general and special interest) meet throughout the world on a regular basis.

Forth Dimensions magazine is published six times a year and addresses the latest Forth news. A one year subscription to FD is free with FIG membership.

The FIG-Tree is the FIG-sponsored, on-line computer data base that offers members a wealth of Forth information. Dial (415) 538-3580 using a modem and type two carriage returns.

Forth publications: a wide variety of high quality and respected Forth-related publications (listings, conference proceedings, tutorials, etc.) are available.

The FIG HOT LINE (415) 962-8653, is fully staffed to help you.

The Job Registry helps match Forth programmers with potential employers.

All this and more for only $15.00/yr. ($27.00 foreign) Just call the FIG HOT LINE or write and become a FIG member (VISA or MC accepted.)

Don't miss our upcoming 6th Annual Forth Convention November 16-17, 1984 at the Hyatt Palo Alto in Palo Alto, CA. Call or write for details.

FORTH Interest Group

(415) 962-8653
PO Box 1105
San Carlos
CA 94070

MICRO Goes FORTH

I agree with the position set forth by Mike Dougherty in the preceeding editorial. Starting with this issue, MICRO intends to provide regular support for programming in FORTH. Our recent Reader Survey indicates that approximately one-fifth of you already program in FORTH. This means that many of you are capable of providing FORTH oriented material for MICRO. The primary thrust should be FORTH programs and utilities that can be added to other reader's 'toolboxes'. Since FORTH is fairly standardized and is available for all microcomputers, the value of each well-written FORTH program extends far beyond the bounds of the microcomputer that it was written on. This should be a refreshing change from machine specific BASIC and assembly programs. If you are a FORTH devotee, here is an opportunity to share your accomplishments and enthusiasm with other serious computerists.

Of course, if one-fifth of the readership program in FORTH, then four-fifths **do not**! For these readers, MICRO would like to provide introductory tutorials, 'how-to-get-started' projects, buyers guides to FORTH materials for specific microcomputers, complete applications and utilities, and overall, an incentive to make the effort to learn a new language. Make no mistake — it does take effort. You will have to purchase a version of FORTH for your system, install it, use an editor that may be totally different from that which you are familiar with, learn a whole new way of approaching and solving problems and memorize a strange, new vocabulary. But, MICRO will be there to help.

The editors at MICRO are not FORTH experts. We are just learning to use FORTH. On the one hand, this means that we will be very sympathetic and understanding about the difficulties other programmers encounter in getting into FORTH. On the other, it means that we will be very dependent on those of you who already are experts to provide the articles and programs that will make FORTH a successful part of MICRO. If you have never tried FORTH, try it. If you are a FORTH enthusiast, support it.

Atari

It was not.suprising to find from the Reader Survey that the two most popular microcomputers were the Apple (39%) and the Commodore 64 (39%), with Atari (13%) a distant third. Still, I feel that the Atari family of microcomputers has many features that should make it interesting to the serious computerist. Unfortunately it has: a 'game orientation' stigma attached to it; a non-Microsoft BASIC with a number of 'odd-ball' constructions — especially in dealing with strings; some annoying aspects such as the 'beeping', the 'graphic mode key' where the right-hand 'shift key' should be, and so forth. MICRO's coverage of the Atari has been weak relative to that of the Apple and Commodore. This is not due to our lack of interest! It is due to a lack of good article submissions on the Atari. Although only about one-third as many readers own Ataris as own either Apples or C64s, I would estimate that we get ten (10) or more submissions for each of these computers to one (1) received for the Atari. We try to convert some articles submitted for other micros to the Atari, (see *Time – Series Forecasting* in this

issue), but this is a lot of extra work. While it might be tempting to just drop the Atari, I still have hope for this micro. Especially now that Jack Tramiel, the man who took Commodore to the top, has taken over Atari. While Atari may currently be down, do not count it out. If those of you who use Ataris start submitting 'meaty' articles, then you will see a lot more in MICRO on the Atari. Let's hear from you.

Advertisers

The money you pay for your copy of MICRO, whether through a subscription or at the counter, does not begin to pay for the cost of producing it. If MICRO was dependent solely on its distribution revenue, then you would be receiving an 8 to 12 page newsletter. The additional revenue required to run MICRO comes primarily from advertising. Advertisers pay to run ads for one reason: they want to make sales. How does an advertiser know if his ad is working in MICRO? While a few have a special 'department number' or other encoded information in their address, most do not. They will only know that you saw their ad in MICRO and were positively affected by it

If You Tell Them!

When you contact an advertiser to buy a product or for more information, please tell them that you saw it in MICRO. You can not over-estimate the effect this will produce. The size of MICRO is determined primarily by the amount of advertising. If you want to see MICRO grow, then

Support Your Advertiser.

Robert M. Tripp

Editor-in-Chief

On The Cover

```
FAMILY
  1____FOUR-GENERATIONS
  2_____FIRST-FAMILY
  3_____GREAT-GRANDPARENTS
  4_____(.") ADAM AND EVE
  3_____GRANDPARENTS
  4_____(.") GRAMMY AND GRANDPA
  2_____NUCLEAR-FAMILY
  3_____PARENTS
  4_____(.") MOM AND DAD
  3_____CHILDREN
  4_____(.") JACK AND JILL
```

For computerists who want to communicate with 'third generation' computers, our tree (a 'structure tree' which was generated by Dougherty's 'Structure Tree Utility') might bring to mind Dr. Moore, who named his solution FORTH (fourth generation language). As Nature creates trees of brilliant variety, so can the user of FORTH.

Cover Photo by Cindy Kocher

New low price $89.95

Telecomputing with a difference!

SuperTerm — the only software that communicates with them all! Information networks such as CompuServe; business and university mainframes; free hobby bulletin boards.

Professionals and students: SuperTerm's VT102 emulation gets you on-line in style. Advanced video features, graphics, full-screen editing, 80/132 column through sidescrolling, extended keyboard — perfect for EDT, DECMail, etc. Even download your workfiles and edit off-line! Full printer and editor support; other emulations available.

Researchers and writers: SuperTerm's built-in text editor helps you create, edit, print, save, send and receive text files — articles, stories, reports, inventories, bibliographies — in short, it's your **information work station.** Access CompuServe, Dow Jones Information Network, Dialog/Knowledge Index, Western Union's Easylink, The Source, and many more. Optional Sprinter accessory saves printing time and $ (see below).

Computer hobbyists: Join in the fun of accessing hundreds of free bulletin board systems (BBS) for Commodore, Apple, TRS-80, etc. Text mode with all BBS systems; up/downloading with Commodore BBS systems (Punter protocol). Special protocol for up/downloading with other SuperTerm owners. Popular "redial-if-busy" feature for use with automodems

Get the information you need, for business or for fun, with **the software that communicates with them all!**

Requires Commodore 64, disk drive, and suitable manual- or auto-modem. Printer optional. Software on disk w/free backup copy. Extensive manual in deluxe binder.

SuperTerm's SPRINTER Accessory $69.95

With the Sprinter accessory, SuperTerm can perform **concurrent printing** — as text appears on your screen, it's simultaneously printed on your printer. Includes all necessary hardware for connecting your **parallel printer** and computer via the cartridge port. Simply plug-in and go. Free utility software for printing and listing as a stand-alone interface.

Requires parallel printer such as Epson, Gemini, Microline, C Itoh. (Min. speed 35 cps.)

Commodore 64 is a trademark of Commodore Electronics, Ltd.

(816) 333-7200 Send for a free brochure.

MIDWEST MICRO inc.

MAIL ORDER: Add $1.50 shipping and handling ($3.50 for C.O.D.); VISA/Mastercard accepted (card# and exp. date). MO residents add 5.625% sales tax. Foreign orders payable U.S.$, U.S. Bank ONLY; add $5 shp/hndlg.

311 WEST 72nd ST. • KANSAS CITY • MO • 64114

feedback

To The Editor

I believe that the most basic need in the micro field is the tranportability of software from one brand of computer to another.

I, unfortunately, purchased a good computer that is no longer very popular. I have converted it to CP/M to relieve the software problem, but find that even that uses a nonstandard disk format which is difficult to get translated from the usual IBM format. Even with CP/M there are about 20 different disk formats.

I believe in innovation but I find it difficult to understand why a program written in a high level language cannot be portable between almost all micro-computers in some reasonably convenient way.

I'm in the process of determining what my next computer will be. But I do not intend to buy the latest model every year. The big question is not what is "state of the art" or what is fastest, it is what computer is the software going to be available for in five year or ten years. Now, most new software is being written for the IBM PC. Even CP/M-80 is being ignored by many vendors. Until recently, I thought that maybe I.B.M. would be good for many years but now I'm almost convinced that technical limitations will prevent if from being effectively expanded to larger systems such as UNIX and ADA. However, I do not think that a Motorola 68000 based system will dominate without the support of at least one large company in a moderate price range. I know that all software writers cannot provide all software for all formats and all languages when almost all computers are different. So why cannot all computers be provided with a second standard ASCII disk format which is common to all computers for the purpose of transporting software and data. Then the addition of software translaters such as Apple Basic to Commodore Basic or even Basic to Pascal could reduce a nasty problem to one that is managable by most users. I doubt that every one is going to agree on any standard by which this can be done, so maybe a solution would be for a technically oriented magazine such as MICRO to publish the software and hardware specs needed to read from the various disk formats.

C. M. Nelson
Indianapolis, IN 46256

The following limerick was submitted by Margic Joseph of Los Angeles, CA.

Though sometimes her memory would slip
And sometimes her mind took a flip,
But now don't dispute'r
She's got a computer
Her memory's a silicon chip.

MICRO

What's Where in the APPLE

- Gives names and locations of various **Integer BASIC, Monitor, Applesoft,** and **DOS** routines and tells what they're used for
- Lists **Peeks, Pokes** and **Calls** in over 2000 memory locations
- Explains how to use the information for easier, better, faster software writing

The revised edition with //e Appendix is now available at a new low price of only $19.95.(plus $2 sh/h).
For the 35,000 people who already own previous editions, the //e Appendix is available separately for just $5.00.

To order, send check or Money Order to:

MICRO INK
P.O.Box 6502
Chelmsford, MA 01824
phone 617/256-3649 *(use VISA or MasterCard)*

McMill 68000 Coprocessor System

Distributor

Stellation Two
P.O. Box 2342
Santa Barbara, CA 93120
(805) 966-1140

Introduction

The McMill 68000 Coprocessor System is designed for the Apple II and IIe. It is a peripheral board that enables the Apple to run 68000 programs. The CPU is a 68008 chip that utilizes direct memory access logic allowing the 6502 and 68008 to alternate memory cycles. All memory and I/O slots can be directly accessed via the 68008. Using alternating cycles while the 68008 is running, the 6502 continues to execute at one-half speed or faster. Special address translation logic resolves conflicts among 68000 exception vectors, 6502 zero page and Apple II I/O space locations. The fact that the McMill 68008 is truly a coprocessor that runs simultaneous tasks with the 6502 is a particularly powerful advantage. (During disk access, game paddle reads and any other timing-loop dependent functions the 68008 must, of course, be halted.)

Installation

The McMill 68008 Coprocessor System board is painless and easy to install. All that is necessary is inserting the board in one of the peripheral card slots. There aren't any additional connections or worries to deal with.

What is Provided

The package includes the McMill coprocessor board, Hardware Documentation Guide, a Motorola Inc. MC 68000 Microprocessor Programming Card, a floppy disc entitled Fig Forth, version 1.0 Mountain View press, hardware warranty (McMill will repair or replace free of charge any board that is defective within one year of the original purchase date, damage caused by accident, misuse, or tampering not included).

You can also order the S-C 68000 Cross Assembler with the McMill Coprocessor System. This is an excellent assembler as those who already have their 6502 version will attest to. All S-C assemblers use the same set of directives and commands, making adaption to different chips very easy. The 68000 version has some differences from the standard S-C Macro Assembler, such as expressions being expanded to 32 bits, and other alterations necessary to accommodate the code and syntax differences in the 68000. All variances are clearly documented. Included in their latest version are 10 new commands and 7 new directives, improving what is already a fine product.

Under a special arrangement with Addison-Wesley Publishing Company Stellation has been allowed to provide a monitor from Tim King and Brian Knight's book 'Programming the M68000.' The book takes the reader through the world of the 68000 and how to program it. A floppy disk with their monitor and a debugger is provided. The monitor/debugger is compatible with the S-C Macro assemblers source files, allowing easy editting and assembly.

What is Not Provided

The board itself hasn't any ROM or RAM. None of the languages most 68000 users would use are provided, namely C, Fortran, UNIX, and LISP. Although a version of C is due to be released, this is still a serious shortcoming.

Documentation

This is where the McMill package is the weakest. The Hardware Documentation Guide could cover more and include a schematic diagram that is legible. Expanded hardware documentation is scheduled for release, which will hopefully correct the shortcomings of the present version. The Monitor/Debugger relies on the information in King and Knight's 'Programming the 68000' for documentation. The information needed is basically there but the organization is not. It is organized as a chapter in a book (which is what it is), not as documentation for software. Although the Mountain View Press Fig FORTH is included with the McMill package, you have to send away to Mountain View Press to get the complete documentation (for a nominal charge). This strikes me as a good way to create unhappy and frustrated users. It would make more sense to charge more for the package and deliver it complete, rather than inconveniencing and possibly annoying the user.

Price

The McMill with FORTH is $229.00; with the S-C Assembler it is $299.00.

Conclusion

This package has its ups and downs. It does gives the Apple user an inexpensive way of upgrading to a 68000 machine, while not having to give up the familiarity of his present machine. On the other hand the poor documentation, lack of on board RAM and ROM, Fortran, LISP and UNIX are enough to discourage many users. For those who are soley interested in getting into 68000 Assembler, using the S-C Macro Assembler with the McMill board is an inexpensive means of doing so. In the end it depends on what your needs and expectations are as to whether this is a product for you.

MICRO

Super Action Software!

PROGRESSIVE PERIPHERALS & SOFTWARE

DENVER, COLORADO

CYBERWORLD $39.95

This five-screen arcade adventure packs the computer with intense graphics and sound! You are a special Cyberleague agent in a universe full of hostile aliens and vicious robots. Joystick and keyboard transport you through 3-D rooms, space barriers, fleets of invaders, and warship-ridden quadrants of space. Over 100 sprites, 8 new character sets, and dozens of mind-boggling sound effects make up this multi-layered adventure. Animation, action, and strategy all combined into a game so extensive that two disk sides are jammed with game programs and data! Reach the ultimate rank of admiral and you may carve a niche in the permanent high-score list. A full-size book quality manual with full-color covers is included to guide you through your most exciting game experience.

NEW

Perplexian Challenger $29.95

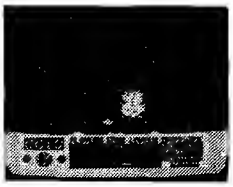
The incredibly responsive three-axis joystick control of a space fighter is in your hands. Split screen graphics provide a continuous display of your ship's instrumentation as well as a three-dimensional animated view of space.

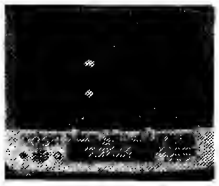
You as a pilot must utilize lightning fast reflexes to destroy invading ships and avoid their return fire. Simultaneously, you must maneuver your ship to capture space debris that remains from the explosions.

Outstanding graphics features include smooth 3-D rotations, split screens, and the most incredible high-resolution hyperspace sequence ever produced.

Programmed entirely in machine language, this action-strategy game is guaranteed to blow you away.

All the professional features you expect are included, automatic self-demo, high score retention, pause and provisions for 1 to 4 players. Add to this features you don't expect like easy-loading and music during the load. Perplexian Challenger is a game that brings the arcade experience to your home.



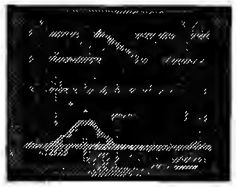


Wizard $39.95

Jump from ropes to ladders, dodge plummeting boulders and duck under deadly arrows in your quest for sparkling diamonds, gleaming bars of gold, and glistening pearls. With joystick in hand you must explore forty dazzling screens, each a new and exciting adventure. Take the key to unlock the doorway to your next spine-tingling level. Each key restores your magical powers, allowing you to cast over ten different spells. With these magic spells you have the power to overcome vicious creatures, terrifying traps, and perilous plunges.

Your Wizard is realistically animated in every possible direction. Dozens of movements are possible — jump over burning fires, shimmy up or down ropes and ladders, even slip down treacherous sliding staircases! Magic portals move your Wizard through midair and protect you from a myriad of fully-animated fiendish monsters. Catch an elevator to the top of the screen and dart through sliding gates in your quest for magic and treasure.

Wizard's fascinating variety of screens are sure to please and entertain, and of course you can build an unlimited number of your own levels using the construction set provided with your game.



NEW

Gothmog's Lair $39.95

Real-time adventure excitement at its best. Solve countless puzzles and slay over a dozen monsters by using the huge vocabulary of over 200 words. Two challenging difficulty levels await you with over 80 areas, each fully described in Old English script.

Menacing monsters, knifing villains, tattered code books and treacherous terrain are just a few of the situations you must overcome in your quest for the thirteen priceless treasures. More than seventy objects are invaluable to you in your search for glory and wealth!

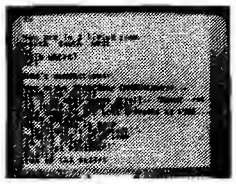
A full-size, thoroughly illustrated manual is included. Featuring color front and back, book quality, and a fold-out map, this extra further extends the professionalism of this game. The following are quotes from unsolicited testimonials sent to us by adventurers in Gothmog's Lair:

I have extremely enjoyed Gothmog's Lair and plan to buy more adventure games.

Scott Tutman
Memphis, TN

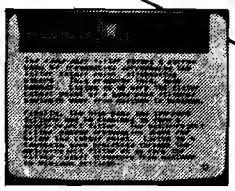
Gothmog's Lair is the best adventure I've ever played.

Dennis Manochio Jr.
Saratoga, CA

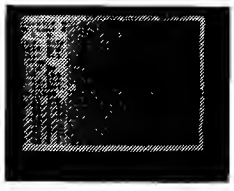


PROFESSOR $34.95

An in-depth self-tutorial for the Commodore 64 on a two-sided disk. This menu-driven tutorial covers every aspect of your C-64—BASIC, keyboard, sound, music, simple and advanced graphics. Quizzes test your comprehension at the end of many lessons! On-screen illustrations, sound effects and full-color interactive graphics make learning easy and interesting. The PROFESSOR is your ONLY choice for an all-in-one, thorough tutorial about the Commodore 64!



Supershipper 64



A complete multi-printer shipping system which operates on the inexpensive Commodore 64. Offers all the features of more expensive, more cumbersome business software at a low, low price. Full-screen, full-cursor editing on all data entry. Prints invoices, C.O.D. tags, mailing and shipping labels. Sorts your customer list alphabetically, by city, state or by salesperson. Keeps complete records of all invoices and accounts on disk — up to 800 accounts per disk! Automatic backups, product charts and many other features are also included.

Supershipper Accounting

The accounting supplement to the Supershipper 64. Prints statements, bank deposits, past due accounts and daily or monthly sales reports. Breaks down sales commissions and prints all customer's past transactions. Also provides inventory control for up to 200 different products. The only way to fully computerize your business!

Supershipper 64	$99.95
Supershipper Accounting	$79.95

Progressive Peripherals & Software
2186 South Holly, Suite #2, Denver, Colorado 80222

VISA

Call for more information or a dealer near you.

MasterCard

TELEX. 888837 (303) 759-5713 (303) 757-0830 TWX. 9109971314

Product Name: **Wizard**
Equip. Req'd: Commodore 64
Price: 39.95
Manufacturer: Progressive Peripherals & Software
2186 South Holly, Suite #2
Denver, CO 80222

Description: If you must have a game, it should at least utilize the built-in capabilities of your computer to the maximum. Wizard does! The game is of the "Donkey Kong" genre in which you move around the screen to reach an ultimate destination, while getting points by taking treasures and avoiding fatal hazards. Through the use of sprites, clever sound effects, a superior collection of options, a large number of screens and many unique concepts, it goes far beyond similar games. For example, there are 'spells' that can be used to ward off calamity. Some of these, such as 'invisibility', can protect you from harm, but at the same time may make it more difficult for you to move around the screen since you can only see the wizard against colored backgrounds. Also, the invisibility 'wears off' as you use it to avoid destruction. High scores are automatically maintained on disk.

Pluses: The utilization of the Commodore 64's sprites, sound, programmable characters and color provide an excellent demonstration of the capabilities of the computer and should inspire programmers to improve their own displays. The game involves more than just the complex coordination found in many other games. A good deal of strategy is required to master the game. The game is even fun to watch while another plays, so that sharing the game in a multi-player mode is enjoyable. The choice of screens, speeds, spells and so forth keep the game from getting repetious, even after many hours of play. The most significant feature is the ability to generate your own screens! This goes beyond merely playing the game. This gives the novice a chance to experience the joy of 'programming' a computer to make it do what he wants. He can design screens as complete as those that come with the package, with hazards, spells, colors and the like.

Minuses: It may become habit forming.

Documentation: Very extensive for a game. Provides more than enough information to play the game and to create new screens.

Skill Level: With the variety of speeds and screen difficulty, the game is suitable for all ages. The new screen creation is limited only by the users imagination, not any knowledge or training limitations.

Reviewer: Robert M. Tripp

Product Name: **Person-to-Person**
Equip. Req'd: Apple II, II+, IIe - DOS 3.3
Modem recommended-Hayes, Apple or Novation
Price: $39.95
Manufacturer: Trutec Software, Inc.
1700 Solano Ave.
Berkeley, CA 94707

Description: A full-featured communications program including a mailing list/telephone list file, auto-dialing, including secondary carriers, terminal program with auto log-on and printed output, including form letters and mailing labels. The entire program is menu-driven with consistancy of response throughout the program. The 80 page documentation is easy to read and completely explains any possible questions. The manual includes many examples of command files.

Pluses: This program has all the bells and whistles of standard communications/ terminal programs, plus it incorporates a complete mailing list program with telephone numbers that can be searched and dialed, including pass word numbers to carriers such as MCI and SPRINT. A single keystroke can dial and log-on to such as CompuServe and Source, upload electronic mail, check your mailbox, download its contents, sign off and hangup.

Minuses: None noted.

Skill level required: No prior experience required.

Reviewer: Phil Daley

Product Name: **Mail Now**
Equip. Req'd: Commodore 64 with disk and printer
Price:
Manufacturer: Cardco, Inc.
313 Mathewson
Wichita, KS 67214
Author: S. & S. Faure and G. Coggin

Description: A user-friendly mailing list program for 600 names per disk with add, delete, modify and sorting options. The printer options include one or more rows of labels, repeat labels, and printer codes for double strike, enhanced, etc. There is also a convert function to change the file format into one readable by the Write Now! word processor.

Pluses: If you are familiar with label systems, the manual is almost superfluous - the program is that easy to follow. The search command quickly and easily can find a string in any major field. Modify quickly changes any data. The program performs flawlessly.

Minuses: If you like slapstick comedy, read the manual. Otherwise just use the program and forget the satire.

Skill level required: No prior knowledge required.

Reviewer: Phil Daley

Product Name: **Stocker1**
Equip. Req'd: Apple II, II+, IIe - CP/M only with Z80 card, TRS80 Mod III, 4, II, 12, 16 card
Price: $300.00
Manufacturer: Engineering Management Consultant
P.O. Box 312
Fairfax, Virginia 22030

Description: This package is designed to help the user forecast stockmarket turning points, enabling better investment and profit performance.

Pluses: Using a unique Moving Window-Spectral method Stocker1 saves time in stockmarket forecasting. Inputting of historical data is easy and user friendly (prompting provided). Stocker1 employs a univariate model (one variable in, the same variable out). Future values are predicted using historical data which is input in a user oriented data editor. Data can be entered in a Create mode or Update mode. Statistical comparison is provided in addition to graphical representation and comparison. Charts/graphs are in an easily readible form. One free hour on the Electronic Forecast Information Service is also included in the package.

Minuses: If the user is not very familiar with stocks and forecasting he is apt to be lost with this program. This is certainly not a program for the uninitiated. At one point through an error entered into the program we ended up in an infinite loop. A second try did not produce this problem. Unless you are a serious player of the market it would be hard to justify the cost of this program.

Documentation: The manual provided is clearly laid out and reasonably easy to read. Instructions and descriptions are understandable if the user has an understanding of the subject. It is assumed the user is familiar with his computer's operating system. Technical data is provided, as are sample case studies, and a bibliography. The manual does suffer from a number of typos.

Skill level: Intermediate to advanced.

Reviewer: Mark S. Morano

Product Name: **MPP-1000C Modem**
Equip. Req'd: Any Atari Computer
Price: $149.95
Manufacturer: Microbits Peripheral Products
225 W. Third Street
Albany, OR 97321
503/967-9075

Description: There probably is not any easier way to get into telecommunications with your Atari. The package consists of the modem which plugs into a joystick port and a standard telephone connector; a cartridge containing the terminal software; and a short manual. The program is totally menu driven, making it simple for anyone to use. The following features are supported: display disk drive 1 directory; direct transfer of information between disk and modem; buffer modem to memory and then copy to a specified device and visa versa; select between full/half duplex, ASCII/ATASCII translations, X-modem protocol on/off, 38/40/80 column display, no/odd/even parity; auto answer; enter/save/load up to ten phone numbers to be used to dial number, and more. By combining the various features, files may be up/down loaded from Atari Bulletin Boards (BBS), other Ataris and other computers. A CompuServe demonstration package is included, and, thoughtfully, a list of bulletin board services listed by state throughout the country so that you have someone to talk to when you first get started.

Pluses: Extremely easy to connect, use, and understand.

Minuses: None noted.

Documentation: Basic information is covered well. A few more examples of combining the various options for specific tasks would have been useful.

Skill level: Any and all.

Reviewer: Robert M. Tripp

OS9
APPLICATION SOFTWARE

ACCOUNTS PAYABLE **$349**

GENERAL LEDGER with CASH JOURNAL **$449**

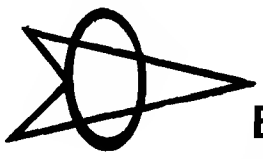
PAYROLL **$549**

ACCOUNTS RECEIVABLE **$349**

SMALL BUSINESS INVENTORY **$349**

COMPLETE DOCUMENTATION $19.95

OS9 & BASIC 09 ARE TRADEMARK OF MICROWARE, INC. & MOTOROLA CORP.



SPECIALTY ELECTRONICS

(405) 233-5564
2110 W. WILLOW – ENID, OK 73701

Introduction to FORTH

Basic why's and wherefore's about the FORTH language.

by Kenneth Butterfield
Los Alamos, New Mexico

In the early spring of 1978, I first learned of FORTH from an advertisement for PetForth by Programma International. At the time, I was doing scientific calculations and was looking for a language that would be faster than the Pet BASIC. Another goal I had for the PET was to use it to control experiments in a physics laboratory. I called Programma to find out if FORTH was suitable for my needs. The call was transferred to the programmer who had written PetForth which really gave me a feeling that I was getting first class service! My first question was *Is FORTH a good Language?* and the response was the classic, *I don't use languages, I just program them!* The question *Is FORTH a good language?* remained unanswered. I hope to provide my answer to this question, as well as to the questions *What is FORTH?*, and *Who would be interested in FORTH?*

First, FORTH is more than just a language. It can be a stand-alone operating system that provides basic support for terminal and disk control. Multi-tasking and multi-user FORTH systems are available. FORTH also provides a block structured high level language that can be used in an interpreted or compiled mode. Among some of the block structures provided are the DO...loop, BEGIN... WHILE...END, and BEGIN...UNTIL loops and the IF...ELSE...ENDIF control structure. On the other hand, FORTH has been called a pseudo-machine language because the key words used for moving data from place to place are very simple and similar in use to the techniques used in assembly language.

FORTH can have all of the above features because it is an extensible language. Most implementations of FORTH have a small kernel written in machine language and the remaining 80-90% of the language is written in FORTH. Any new 'words' (FORTH's name for a procedure or subroutine) are defined using the previously compiled words. Each new word becomes part of the FORTH 'dictionary' (list of words or commands) and are available for use in future definitions. The programs that you write become part of the language.

One of the nicest features available in FORTH is the on-line interpreter. Commands may be given to FORTH from the keyboard in a similar manner to the 'immediate mode' of most BASIC interpreters. This allows FORTH to be used as a calculator. Another use for the interpreter is in program development and debugging. The interpreter allows the programmer

to try out a sequence of commands, one at a time, to verify their consequences. After the sequence has been shown to work properly, it can be given a name and compiled into the language for future use. A program (word) is activated by typing its name in the interpreter mode, or by entering the name in the list of names that makes up the definition of a new word.

FORTH was originally written by one man, Charles Moore, to provide a better media for program development than the languages available at the time. One of its first uses was in real time computer graphics where, it is said, FORTH provided a marked increase in speed over FORTRAN(1). The first widespread use of FORTH was in the computer control of large telescopes, and FORTH continues to be the language used at many of the world's largest observatories. In fact, my early interest in the language grew when a fellow student informed me that FORTH was the language he used when doing his research at the Kitt Peak Observatory. It seemed to me that a language that could control a large telescope should be useful in other control applications.

More recent uses of FORTH continue the computer control theme. Two applications are the control of robot cameras for special effects in the motion picture industry(2), and remote sensing of water depth and speed for aid in navigation of large barges on inland waterways(3). FORTH is still strong in the area of computer graphics. Charles Moore has a CAD-CAM system that runs in 28K of memory(4). FORTH has been shown to be very useful in research laboratory settings where the experiments vary from day to day. It is important to have a language that is flexible, allowing the measurement (spectrometers, ADC's, etc.) and control devices (stepping motors or relays) to be connected in new and often changing configurations.

New uses for FORTH are being developed all of the time, and some of the most exciting are in the area of artificial intelligence. LISP is the language usually associated with AI projects, but there is a marked similarity in the underlying structures of LISP and FORTH. Both languages treat data and programs as lists, and have the feature that a program can create and manipulate a list that will later be used as a program. Programs written for the two languages are similar except for the notation. LISP uses a parenthetical notation while FORTH uses a parenthesis free notation originally derived for formal logic by a Polish logician, Lukasiewicz. The difference in usage can be compared to the difference in using an algebraic calculator such as produced by Texas Instruments, versus a 'Reverse Polish' calculator from Hewlett Packard

In the FORTH compiler, new words are added to the dictionary, and can be used in the definition of future programs. The structure of a word consists of a header section containing the name of the word and pointer and a data section. The data can be either a constant, a variable, a list of variables, or a list of addresses if the word is an executable word. It is LISP's ability to treat a list as either data or program that makes it useful for AI programming. It is not surprising, then, that FORTH can also be used for AI applications. What is surprising is that FORTH may have significant advantages for some projects.

For instance, in a knowledge-based system developed by General Electric to diagnose and troubleshoot large electric locomotives, a FORTH system operating on a PDP-11 minicomputer was found to be smaller and faster then a LISP system operating on a VAX computer(2). If you think that such a system would be out of your reach, a program recently became available for CP/M based computers that supplies an expert system language written in FORTH. A recent article uses this language to develop a weather prediction program(5).

Hopefully, I have convinced you that FORTH is a powerful language that is useful for many different applications. Not only has FORTH been used to control large telescopes and explore artificial intelligence, it has been used to program text editors, data base systems, spreadsheets and, of course, games. Application programs of these types are often available from the same vendor that sells a version of FORTH for your machine.

If you are interested in FORTH there are many good sources of information. Various computer magazines print articles on FORTH either occasionally or on an annual basis. Other sources of information include newsletters and books.

MICRO has told me that it plans to publish more FORTH articles now that their emphasis has returned to reaching the experienced users. Dr. Dobbs Journal has had an annual FORTH issue for several years. It comes out in September and is usually very good. Several other magazines have published an article on FORTH at one time or another. You might check the availability of back issues.

One of the best information sources is the FORTH INTEREST GROUP, FIG, (PO Box 1105 San Carlos, CA.94070). FIG publishes a bi-monthly newsletter and has local chapters in many cities. Membership in FIG is $15 and includes a subscription to FORTH Dimensions. FIG also has source listings of FORTH for many microprocessors and many minicomputers. The FIG-FORTH installation manual is a very interesting document and I highly recommend its purchase because it contains vocabulary listings with descriptions of the use of each word. It also contains a complete FORTH implementation written in FORTH as an example of how large programs can be written!

There are several good books available ranging from introductory level to advanced application. Starting FORTH by Brodie is a very good introductory text with the caveat that it uses FORTH Inc. syntax. Threaded Interpretive Languages by Loeliger is a very good text on the design and implementation of threaded interpreters (FORTH is only one example) and would be a good choice for more experienced computerists. Mountain View Press, Inc. (PO BOX 4656 Mountain View, CA. 94040) lists over 30 books and manuals, a dozen of these being general descriptions of FORTH.

One last source that should not be overlooked is the instruction manual that comes with a FORTH system. There are many vendors that supply FORTH systems, and many of these have very good documentation that comes with their product.

This brings up my final topic. If you decide to get a FORTH system, what should you buy? As usual, a simple question like this cannot be answered simply. First you have to decide whether to implement your own version of FORTH using a source listing (from FIG) or to purchase a complete system. There are complete systems for practically every computer that has been on the market for more

than a few months. In fact, one computer (Jupiter Ace at $150) comes with FORTH as its main language. If you decide on a complete system you then have to decide on which FORTH standard to use.

Most people will want to purchase a complete system from a vendor. This has the advantage of having a working program immediately, and someone to complain to when it doesn't function in the expected manner. The main standards that exist are original FIG-FORTH, FORTH79, and FORTH83. These are all standards defined by FIG. Fortunately, FORTH83 supercedes FORTH79 and should become the most common standard as vendors update their product. Original FIG-FORTH is very common and can be extended to meet the later standards.

Another version of FORTH comes from FORTH Inc. This company was started by Charles Moore, so their FORTH has to be considered as a standard because it is the 'original' FORTH. FORTH Inc. produces a very professional package for many different computers. They might be considered the Rolls Royce of FORTH systems.

There are many vendors that produce a FORTH that does not meet any of the above standards for one reason or another. Quite often these systems have a marked speed improvement over a standard system. Be aware, however, that the speed increase is usually obtained at the sacrifice of portability to other machines (including other processors) which is one of the main benefits of the standard systems. Another possible sacrifice that is sometimes made is the ability to produce ROMable code. Of course this capability may not be something you need. Then too, this ability may not be an option on a standard system.

My recommendations for personal use would be to buy a FIG-FORTH compatible system, and preferably one that has enhancements to meet FORTH83 standards. This type of system can run programs written for many different computers. Your first system should probably be a complete system purchased from a vendor. If you are really brave or foolhardy you might consider the option of configuring your own system starting with FIG source code. I have done this for a 6809 computer and succeeded (mostly) in getting it to work. This is a particularly good route for someone who has built a computer from scratch. Implementing your own system is not easy, but it is tremendously satisfying when completed! In any case, FORTH is a very interesting language that is well worth learning.

References

1) Marlin Ouverson, Interview with Charles Moore, FORTH Dimensions vol. 6, 2, pg. 2, July/Aug. 1983.

2) Kim Harris, Forth Applications Conference, FORTH Dimensions vol. 6, 2, pg. 31, July/Aug. 1983.

3) Peter J.Largeren, FORTH: Cheaper than Hardware, FORTH Dimensions vol. 6, 2, pg. 13, July/Aug. 1983.

4) Robert Berkey, FORMAL 1983: A Review, Programming Techniques, FORTH Dimensions vol. 5, 5, pg. 34, Jan./Feb. 1984.

5) Jack Park, Expert Systems and the Weather, Dr. Dobbs Journal vol. 9, 4, pg. 24, April 1984.

MICRO™

C64-FORTH/79
New and Improved for the Commodore 64

C64-Forth/79™ for the Commodore 64-$99.95

- New and improved FORTH-79 implementation with extensions.
- Extension package including lines, circles, scaling, windowing, mixed high res-character graphics and sprite graphics.
- Fully compatible floating point package including arithmetic, relational, logical and transcendental functions.
- String extensions including LEFT$, RIGHT$, and MID$.
- Full feature screen editor and macro assembler.
- Compatible with VIC peripherals including disks, data set, modem, printer and cartridge.
- Expanded 167 page manual with examples and application screens.
- "SAVE TURNKEY" normally allows application program distribution without licensing or royalties.

(Commodore 64 is a trademark of Commodore)

TO ORDER
-Disk only.
-Check, money order, bank card, COD's add $1.65
-Add $4.00 postage and handling in USA and Canada
-Mass. orders add 5% sales tax
-Foreign orders add 20% shipping and handling
-Dealer inquiries welcome

PERFORMANCE MICRO PRODUCTS

MasterCard

770 Dedham Street
Canton, MA 02021
(617) 828-1209

VISA

ATTENTION COMMODORE 64 OWNERS:

"Is **THE CLONE MACHINE** really dead?"

Yes, there comes a time when a product grows old and isn't the latest state of the art. Thank goodness we understand that here at Micro-W. Our all new version (known as **SUPER CLONE**) will surely prove that we are still number one in the back-up business.

You'll still get the old reliable Clone Machine but we've added the following:
1) A fast clone copy (approx. 14 minutes) that's simple to use
2) A Super Unguard utilility that quickly handles errors 20 thru 29 (and you don't even have to disassemble your drive like some of our competitors suggest)
3) A new unique way to back-up formerly uncopyable software.

Don't worry if you are a registered owner of our earlier version, we've got you on file and this upgrade will only cost you $10 plus shipping and handling. Dealers, call us for stock balancing on old merchandise

STILL ONLY $49.95*

Should've made back-ups with Super Clone

MasterCard VISA Available from:

Micro-W. DISTRIBUTING, INC

1342B Route 23
Butler, N.J. 07405
CALL: (201) 838-9027 To Order
Dealer and Distributor Inquiries Invited

* We will allow $15 trade credit for any other copy program that you have purchased toward the purchase of **SUPER CLONE** at $49.95. You must provide your original purchased product and state why you want ours instead. This offer may be withdrawn at any time.

Multi-Tasking in FORTH

by Kenneth Butterfield
Los Alamos, New Mexico

SUMMARY

Multi-tasking is a method of allowing your computer to work on more than one program (task) at a time. I present a program that allows two separate tasks to run in the 'background' while still having the FORTH interpreter available in the 'foreground'. The sample tasks are a clock display, and an animated bouncing ball. These can be replaced by just about anything, including separate control programs for 'software robots'. I have given enough detail to allow you to develop other uses of multi-tasking.

A technique and program for running multiple tasks under FORTH.

FORTH is more than a language; it is an operating system as well. In addition, FORTH has the advantage of being changeable, and extendable. This makes FORTH an ideal media for learning how to implement various system operations. This paper will show how to implement one type of multi-tasking system. It includes a demonstration program comprised of two separate tasks (programs) that display the time, and a bouncing ball on the screen while FORTH is waiting for input from the keyboard.

To run this multi-task system you will need a fig-FORTH system with an assembler. The multi-task words will work on any 6502 based machine with modification being required only in the stack partitioning (see section on extensions). Stack partitioning is defined in **RESET.POINTERS**, and I have shown partitions for two FORTH systems. The demonstration is written for either PET or CBM machines with version 3 or 4 BASIC. It should be easy to modify most parts of the demonstration to work on other computers.

For those who are very thorough proofreaders, the demonstration can be set up using screens 110, 111, 115, 116, and 117. Screens 112, and 113 are useful for debugging. Be sure the assembler screens for your system are loaded and then load screen 117. The other screens will be loaded automatically. As compilation takes place you will be told that some words (**KEY**, **EXPECT**) are not unique. When the prompt appears, run **START**. If everything is correct you will have a clock and a bouncing ball displayed on the top of the screen. In addition, FORTH will be ready for input from the keyboard. Hit a RETURN and you should get an 'OK' response. Try adding two numbers and printing the result. While you are typing, the display will operate normally. However, when you hit return it will stop momentarily while FORTH interprets the line, and then it will start up again. Now enter the following line.

```
: TEST 100 0 DO I . LOOP ; TEST
```

Note that the display is halted until **TEST** completes. Next enter

```
: TEST1 100 0 DO I SLEEP . LOOP ; TEST1
```

This time the display should continue while the numbers are being printed. The lesson here is that you can define any FORTH word so that the multi-tasking continues simply by inserting sleep anywhere inside the most active loop.

If you are the cautious type, or if the demonstration didn't work, you will want to debug each module separately. Screen 112 contains a simple example of multi-tasking that can be used to test screens 110 and 111. If you don't have a Pet computer this screen will be the easiest way to try multi-tasking, since it won't require modifying the demonstration screens to run on your machine. Each of the tasks in the demonstration can be tested in a regular (single-task) FORTH by removing the **SLEEP** in **KEY**, **BB**, and **CLOCK**. Once all of the pieces are working properly, go back one paragraph and try again.

A few words are in order regarding the new input structure used in the multi-tasking demonstration. It is based on the FORTH model and stores every key (including cursor keys). Both the back cursor and delete keys will

back up one space. All other cursor keys are entered as part of the input. When FORTH tries to interpret the line you will get errors for any cursor key. If you want to change the way FORTH reads in a line all you need to do is change the definition of **EXPECT**. This is how **PET INPUT** works in Cargil and Riley's FORTH. Incidently, you can modify **PET INPUT** by adding a **SLEEP** in one of the inner loops, and have much nicer input for the multi-tasking system.

TYPES OF MULTI-TASKING

There are many techniques that can be used to implement multi-tasking programs. These range from being straightforward to being involved and complex. Most multi-tasking techniques are simple in theory regardless of how difficult they are to implement. I will describe a couple of methods, then show how to program one method using FORTH.

The most familiar method of performing a series of tasks is to do them sequentially. In FORTH this is done by defining a word for each task. Next, these words are invoked in order, either from the keyboard or from another word. In BASIC the tasks would be a set of subroutines and a main program would be used to call the subroutines in order. This method of multi-tasking is sometimes called the *hen – and – piglets* method [1].

The hen-and-piglets method is fine for simple programs. It requires no programming overhead. It is simple to understand, and it is easy to add another routine into the loop. The main problem with this method is that a slow task must finish before any other task can start. How many times have you waited for a long printout? Another problem is that it might be hard to determine the location for the RETURN required for each subroutine. For instance, in a terminal emulator program, getting characters from the keyboard and from the modem are separate tasks. Sending characters to the CRT and to the modem are other tasks. Or are they? Characters are sent to the modem only when received from the keyboard. Should the task be to get a character and send it? What if no character is present? In this case the task boundary becomes confused if a RETURN is required.

Another type of multi-tasking is timesharing. For users of big computers, timesharing is a well known, and often cursed, way of life. Timesharing has advantages, but it is complex. I am prejudiced against timesharing because it is almost synonymous with multi-user systems. I bought a computer to have it to myself. On the other hand, timesharing offers advantages to a single user who can run more than one task at a time. An example of this is *spooling* of printer output. Spooling means writing the listing to a disk file (fast), and then copying the file to the printer (slow) using a separate task. This task will share time with all other scheduled tasks making it look as though several things are being done at the same time.

Timesharing is complex because the time to switch tasks is usually determined by a clock that interrupts the CPU. The next task to run is determined from a list of tasks and their priorities. This requires some calculation and adds to the programming overhead. Fortunately, there are other methods of multi-tasking that can also have tasks run at what appears to be the same time.

SLEEP

The method I am going to implement is called SLEEP. SLEEP resembles the hen-and piglets method, but has some important differences, the main difference being that each routine is a closed loop. Instead of a RETURN at the end of the task, a call to SLEEP is inserted anywhere in the closed loop of the routine. SLEEP then changes the current task to the next task in a list. In a sense SLEEP plays the part of the main control loop in the hen-and-piglets method.

As in the hen-and-piglets method, tasks are changed under simple and direct programmer control. There are no automatic switchings of tasks at unknown times and places as might happen in a timeshared system. There is also no need to write interrupt handling routines. The order of the tasks which will be run is determined by SLEEP. There is no need to calculate priorities at the time of a switch so overhead is kept to a minimum.

Unlike the hen-and-piglets method, tasks do not have to be completed sequentially. SLEEP can multi-task a series of routines so that they appear to run simultaneously. This is possible because the call to sleep does not have to be at the end of the routine. If it is placed inside of a loop (say at a point that requires waiting for I/O), then only one pass through the loop will be completed before another task is given time to run. Note that the call to sleep can be anywhere in the loop. It can be at the start, the end, or anywhere in the middle. The programmer can chose a place that makes sense to him. When SLEEP next calls (awakens) the routine, the loop will continue from where it left off. SLEEP's ability to run tasks simultaneously is similar to time-sharing, so spooling is one of its possible uses.

One of the main advantages of SLEEP over hen-and-piglets is that each task can have its own stacks and variable storage. If each task has separate storage for data stack, it will not interfere with any of the the other tasks! In fact, two tasks may be the same program. It is not necessary to have separate copies of the programs for each task. Only the data storage needs to be separate. Of course, tasks can interact with each other, when desired, through the use of shared memory.

IMPLEMENTATION OF SLEEP

There are several general considerations that need to be kept in mind when implementing a SLEEP system. Each task should either be an endless loop, or have some way of removing itself from the list of tasks. There needs to be a way to call SLEEP. There needs to be a way to set up new tasks. And last, there needs to be a way to switch stacks, variable memory, and program control.

For the sample multi-tasking system implemented here, each of the tasks will be an endless loop. This keeps the system easier to understand by removing spurious FORTH words. Many problems will require that the tasks be endless loops, so this system is useful as it stands. For instance, in the terminal emulator program mentioned earlier, the tasks that monitor the keyboard and modem continuously check for receipt of characters. When a character is received from either the keyboard or the modem, it is stored in the appropriate buffer. A third task will check to see if the modem buffer has

anything in it. If there is a character, it will be written to the screen. Then the task will loop back to the start. A fourth task would monitor the keyboard buffer and send characters to the modem. All four tasks would be endless loops. They would be written and debugged independently of each other, simplifying the programming effort. In each of the four tasks, a single call to SLEEP would be added somewhere in the loop to tie the system together.

FORTH provides a simple way to call sleep. All that needs to be done is to compile the name **SLEEP** somewhere within the main loop of the task. This can be done directly or indirectly. Because one of the better places to put **SLEEP** is at the end of a loop, one way to compile **SLEEP** indirectly is to redefine **LOOP**, **REPEAT**, **UNTIL**, etc. This makes programming look identical to a non-SLEEP environment. One problem with this approach is that more than one **SLEEP** may be compiled. This makes a task sleep more times then necessary. The task will still run but at a slower pace and with more overhead for the SLEEP program. Even though indirect compilation of SLEEP can be slower, I generally use it because it is easier. Words that work in a normal FORTH environment need only to be reloaded using the new loop words to work with **SLEEP**.

The initialization of the FORTH SLEEP system has several steps. Tasks must be added to the list of tasks to run. Separate stacks (and variable storage areas) must be set up. The return stacks for each of the tasks must be initialized. The last step is to start the system. In the example given on screen 112 these steps are done by running **NEW.TASKS** and **MAIN**. **NEW.TASKS** uses **RESET.POINTERS** to create separate return and data stacks for each task, and uses **INIT.TASK** to initialize the return stack. **MAIN** starts the multi-tasking system after it has been initialized. It also serves as the list of tasks to run.

MAIN, the main control word, awakens four tasks over and over until the break or stop key is pressed. Each task prints its number and then puts itself to sleep. When it is reawakened, it executes the **AGAIN** to loop back. Thus, the output is a list of the task numbers printed repeatedly in order.

In the example, the data and return stacks have been partitioned into five pieces by **RESET.POINTERS**. One piece is for each of four slave tasks and one is for the main task. **INIT.TASK** pushes the parameter field address (PFA) of the word acting as a task onto the appropriate location in the return stack. The data stack needs no initialization.

The key to changing tasks is the word **SWITCH**. **SWITCH** saves the current stack pointers, resets the stack pointers for the next task, and transfers control. The actual transfer is performed by the FORTH word **;S**.

In ordinary operation, **;S** is compiled into a colon definition by **;**. Hence, it serves as a return from subroutine. **;S** pops an address off the return stack and places it in the IP (instruction pointer) register. **;S** finishes with a jump to **NEXT** which loads the W register from the address in the IP. **NEXT** concludes by jumping indirectly via the W register to machine language code. (That's three levels of indirection in **;S**.)

The words **TS1**, **TS2**, **TS3**, and **TS4** are required for **SWITCH** to work properly. They push the task number on the data stack and, more importantly, set up the return stack for the eventual return to **MAIN**. The return stack preparation is devious. The **:** used to define **TS1**, etc., compiles the word **DOCOL**. **DOCOL** is FORTH's jump to subroutine. It pushes the current value of the IP onto the return stack. The **;** (actually **;S**) of **TS1** is never executed. After the stacks have been switched, the jump to **;S** at the end of SWITCH is performed instead. This passes control to a new task (TASK1). The new task will execute until it reaches **SLEEP**. The stacks are switched again, and finaly the return address pushed to the return stack by **TS1** is loaded into the IP. Control is now returned to **MAIN**. Hence, the **SWITCH** located in **SLEEP** serves as the **;** for **TS1** and vice versa. Similar actions occur for **TS2**, **TS3** and **TS4**.

The initialization of the return stack can now be understood. (See definition of **INIT.TASK**.) **TASK1** will put the PFA on the data stack. The PFA is then stored on the return stack, ready to be used by **SWITCH**. The array RP contains the initial values for each of the separate return stack pointers. The return stack is located in page one of memory. The stack grows toward low memory, and the stack pointer points to the next available address. (See Figure 1.) Thus $0101 is added to the value obtained from RP when the storage address for initializing the return stack is calculated. The PFA of the task is used because it points to the list of code field addresses (CFA's) of the words comprising the task. Hence, it is the address of an address that points to machine code. Counting the return stack pointer, the three levels of indirection needed for **;S** are now evident. *[A more complete discussion of the FORTH inner interpreter is contained in reference 4].*

address	value	return stack pointer
Ø1BF	...	
Ø1BE	...	
Ø1BD	PFA HI	
Ø1BC	PFA LO	
Ø1BB	...	<== RP=BB

Figure 1. Initial return stack#1 where PFA is the parameter field address of the task that will be TASK1.

EXTENSIONS OF THE EXAMPLE

There are several points to consider when using **SLEEP** with a more complex set of tasks. These points include determining the size of the stacks and the technique used to store the list of tasks to be switched. You will also need to consider what types of FORTH words are suitable for multi-tasking and where to place any variable torage.

An example of where things can go wrong can be easily demonstrated. While the demonstration (screen 117) is operating define the following word.

```
: BALL 4 12 300 Ø DO MOVE POS DRAW SLEEP
LOOP DROP DROP ;
```

This **BALL** looks like **BB** except that it runs for only 300 times. When you run it, however, it doesn't act at all the same. In fact, the original ball is also acting very strange. What went wrong? The problem is that both **BB** and **BALL** use the variables DX and DY. When one of them needs to change direction, the direction for both is changed. This is an example of tasks sharing data when they shouldn't. To fix this

problem either **BALL** should be redefined starting from the definition of DX and DY, or DX and DY should be stored on the stack instead of in variables. Since each tasks has its own stack, there will be no interaction.

In the 6502 version of FORTH, the data stack and return stacks are limited in the amount of memory that they have available. The FORTH return stack uses the 6502 stack pointer. This limits the size of the stack to one page of memory. In addition, The terminal input buffer (TIB) uses some of the same space. Therefore, it is very important to estimate the size for each of the tasks to be multi-tasked and to partition the stack memory accordingly. Two bytes of the return stack are used for each level of nesting of FORTH words. In addition, each active **DO LOOP** requires another four bytes, two bytes for the index counter, and two for the counter limit.

The data stack has even less space available then the return stack. FORTH sets aside part of the zero page of memory for this stack and uses the 6502 X register as a stack pointer. Zero page is also used for the IP W, and other FORTH registers. To make matters worse, the PET uses half of zero page for the operating system. Other computers may also reserve zero page storage.

The exact location of the data stack limits will vary from one vendor's version to another. For FULLFORTH, the stack is from 4 to 6E. For FORTH by Cargil and Riley, the stack is from 20 to 88. S0 can be used to find the start (high limit) of the data stack. S0 is a user variable that is normally used during an **ABORT** to set the data stack pointer. The lower limit is a little harder to find. You will have to decompile the word **?STACK** [2]. Most versions of FORTH come with a word to perform this function. If your version does not, you will have to do it by hand.

If either of the stacks is not long enough for the multi-tasking system you are implementing, then you will have to set up separate stack storage areas for each task. **SWITCH** will need to be modified to copy the various stacks into and back out of the separate storage areas. If care is taken in the rewriting of **SWITCH**, only the currently used parts of the stack will be copied. This care will minimize the time needed to switch from one task to another. The advantage of stack swapping is that each task can have a full sized data and return stack. The disadvantage is that the overhead time (the time spent while not doing the required tasks) is larger than for partitioned stacks.

Another consideration when implementing multi-tasking is the method which is used for saving the list of tasks to be executed. **MAIN** serves this function in the current example. Using the technique illustrated by **MAIN**, it is possible to change tasks at any time, even while multi-tasking is going on. However, there will always be the same number of active tasks. If you only want three tasks, **NUL.TASK** can used to replace one of the active tasks. The price paid will be the time spent switching to and from **NUL.TASK**. The following line can be used to replace task 2 by **NUL.TASK**.

```
' NUL.TASK RP 2+C@ 1Ø1+!
```

Any of the other tasks can be similarly replaced simply by changing the '2' to the proper number.

If it is absolutely required that the number of tasks be variable, some other technique must be developed to switch from one task to another. Screen 113 is one way that it can be done. The variable NUM.TASK is used to store the current number of tasks. The constant MAX.TASKS is the maximum number of allowed tasks. A new word **TASK.SWITCH** replaces **TS1**, **TS2**, **TS3** and **TS4**. **SLEEP** will still be used by each of the tasks to return to **NEW.MAIN**. **NEW.MAIN** uses a **DO LOOP** index to determine which task to execute next. When all of the current tasks have been called once, it loops back to the **BEGIN**. NUM.TASK is used to set the number of active tasks for this set of calls.

This new technique allows the adding or subtracting of tasks, but the initialization and task removal problems are more complicated than with the original system. You will need to consider what to do when task 2 is to be removed while task numbers 3 and 4 are still required. There are also problems associated with adding a new task to a list of tasks that have been in operation for a period of time. In this case only the new task's stacks should be initialized. If all of the stacks were initialized, each would 'forget' everything that it had already accomplished. In many situations that could be a disaster. I have not given any listings for solving these problems because the solutions depend upon whether stack partitioning or stack switching is being used.

With either **MAIN** or **NEW.MAIN** there are some important considerations for the tasks themselves. First, tasks must be colon definitions. Tasks should either be endless loops (**BEGIN**...**AGAIN**), or provisions for termination of the task will be required. Each task must call **SLEEP** somewhere within the main loop of the task. The last consideration has to do with the storage of any variables used by the separate tasks.

The best place to store variables is on the data stack. If a FORTH variable is used for storage, and more than one task uses that variable, there is a good possibility for confusion. This problem can be formally stated as: *tasks used in a multi-tasking system have to be reentrant.* Reentrant means that two versions of the same task can run at the same time (but not necessarily in synchronization). It isn't always possible to use the data stack, and there are at least two solutions for variable storage provided by FORTH.

When only a small number of variables are required, defining USER variables is a good solution. A user variable is used just like a regular variable. The advantage is that USER variables are addressed indirectly through the user pointer (UP). The UP can be changed for each task to point to separate user table areas. Of course these areas must be set aside and protected. If USER variables are used then **SWITCH** should be expanded to include switching the UP. 6502 FORTH has 128 bytes set aside for the original user table, and the first 48 variables are pre-defined as system variables. These system variables will need to copied into each of the user tables during initialization.

If your application needs more storage than is available in USER variables, separate vocabularies for each task might be a solution. This

```
SCR # 110
 0 CR ." TASK SWITCHING ROUTINES " BASE @ HEX
 1 ( STORAGE FOR TASK STACK POINTERS )
 2 ( ROOM FOR 5 BYTES )
 3 0 VARIABLE RP 3 ALLOT     0 VARIABLE SP 3 ALLOT
 4 0 VARIABLE TASK# ( CURRENT ACTIVE TASK )
 5 CODE SWITCH ( INDEX... ) ( SWITCHES TO NEW TASK )
 6         ( GET OLD AND NEW TASK NUMBERS )
 7      TASK# LDY, 0 ,X LDA, TASK# STA, INX, INX,
 8         ( SAVE PRESENT POINTERS )
 9      TXA, SP ,Y STA, TSX, TXA, RP ,Y STA,
10         ( SET NEW POINTERS )
11      TASK# LDY, RP ,Y LDA, TAX, TXS, SP ,Y LDA, TAX,
12         ( SWITCH TO NEW TASK )
13      ' ;S JMP,
14 BASE ! -->


SCR # 111
 0 CR ." TASK SWITCH--2 "   BASE @  HEX
 1 : SLEEP 0 SWITCH ;
 2 : TS1 1 SWITCH ; ( SWITCH TO TASK 1 )
 3 : TS2 2 SWITCH ; ( SWITCH TO TASK 2 )
 4 : TS3 3 SWITCH ; ( SWITCH TO TASK 3 )
 5 : TS4 4 SWITCH ; ( SWITCH TO TASK 4 )
 6 : NUL.TASK BEGIN SLEEP AGAIN ;
 7 ( TASK INITIALIZATION WORDS )
 8 : RESET.POINTERS   ( INIT. POINTERS )
 9     E8 1 RP + C! 8B 2 RP + C! 6B 3 RP + C! 54 4 RP + C!
10     80 1 SP + C! 50 2 SP + C! 40 3 SP + C! 30 4 SP + C!
11 ; ( 60 1 SP ... 30 ...     20 ...    10 IS FOR FULLFORTH+ )
12 : INIT.TASK ( ADDR, INDEX....)       ( INDEX IS TASK NUMBER )
13     RP + C@ 101 + SWAP OVER !        ( ADDR IS CFA OF WORD )
14     ' SLEEP CFA SWAP 2 + !           ( NEEDED FOR FIRST RUN )
15 : BASE ! ;S


SCR # 112
 0 CR ." MULTI-TASKS FOR FORTH " BASE @ DECIMAL
 1 : MAIN BEGIN ." MAIN" TS1 TS2 TS3 TS4 ?TERMINAL UNTIL ;
 2 : TASK1 BEGIN CR ." TASK 1 " SLEEP AGAIN ;
 3 : TASK2 BEGIN CR ." TASK 2 " SLEEP AGAIN ;
 4 : TASK3 BEGIN CR ." TASK 3 " SLEEP AGAIN ;
 5 : TASK4 BEGIN CR ." TASK 4 " SLEEP AGAIN ;
 6 : NEW.TASKS ( INITIALIZE STACKS AND RUN MAIN RUN LOOP )
 7   RESET.POINTERS
 8   ' TASK1  1 INIT.TASK
 9   ' TASK2  2 INIT.TASK
10   ' TASK3  3 INIT.TASK
11   ' TASK4  4 INIT.TASK
12   MAIN
13   ." ALL DONE "
14 ;
15 BASE ! ( RESTORE BASE ) ;S


SCR # 113
 0 CR ." NEW MAIN LOOP ROUTINE "
 1 0 VARIABLE NUM.TASK (CURRENT NUMBER OF ACTIVE TASKS )
 2 4 CONSTANT MAX.TASK ( MAXIMUM NUMBER OF TASKS )
 3 : TASK.SWITCH SWITCH ; ( REQUIRED TO SUPPLY ; )
 4 : NEW.MAIN
 5      BEGIN NUM.TASK @ 1+ 1 DO I I . TASK.SWITCH LOOP
 6      ?TERMINAL UNTIL
 7 ;
 8 ;S
```

allows for the use of standard variables and arrays. Each task, with all of its storage variables and arrays, will have to be compiled into a vocabulary. If the same FORTH word will be used for two of the tasks, then it will have to be loaded twice, once into each vocabulary. Vocabularies offer as much storage as you have memory. The only disadvantage is that the PFA's of words that will be tasks may be a little more difficult to obtain. Thus, the initialization process may be more complicated then in the example.

CONCLUSION

The answer to the question *What are the objectives of a multi-tasking system!* will depend on who is asking the question. Most people would agree that the first two objectives are to switch between tasks (programs) and to avoid having tasks interfere with each other. Two other objectives, mutually exclusive, are to make the task switching transparent to the user and to optimize input/output operations. There is a set of implementation requirements that will have to be satisfied for whatever set of objectives is chosen. I have covered many of these requirements in the text, but an explicit listing of them might be in order. Multi-tasking requires a method of:

1) initializing the system

2) maintaining separate task data storage

3) initiating the task switching

4) adding new tasks

5) terminating tasks that have been completed.

All of this must be done in a way that minimizes the time required. After all, multi-tasking is useless if the tasks never get a chance to run.

The FORTH program described in this paper describes how to implement the type of multi-tasking called **SLEEP**. The key word in this program is **SWITCH**. **SWITCH** trades the system and task variables associated with the old task for those variables associated with the new task. Because of its importance, **SWITCH** has been coded in machine language. All of the other

FORTH words are high level definitions. Very little time penalty is incurred for the high level words because these words are used infrequently. The balance between machine language and FORTH words in this program was chosen to illustrate the point that the speed of operation of a program can often be improved when only a small piece is written in machine language. *[Since the initial submission of this article, two related articles on multitasking have appeared. See references 5 and 6.]*

REFERENCES

1) *A Simple Implementation of Multi-tasking*, by Wendell Brown, BYTE, Vol 6, 10, Oct. 1981, pg 176

2) *fig-FORTH 6502 Assembly Listing*, by W.F. Ragsdale, Sept. 1980, Forth Interest Group, PO Box 1105, San Carlos, Ca. 94070

3) *fig-FORTH Installation Manual*, by W.F. Ragsdale, Nov. 1980

4) *fig-FORTH Interpreters*, by C. H. Ting, FORTH Dimensions Vol 6, #1, May/June 1984, pg 12

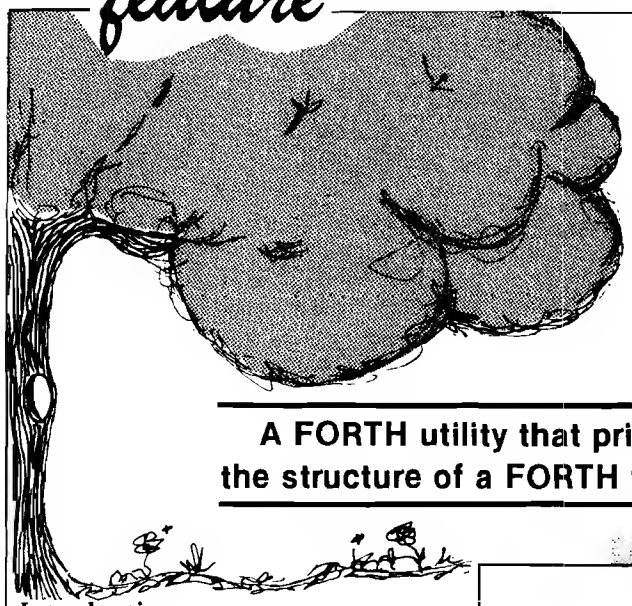
5) *A Simple Multitasker*, by Ray Duncan, FORTH Dimensions, Vol 5, #2, July/August 1983, pg 20

6) *A Simple Multitasking Environment*, by Martin B. Perti, FORTH Dimensions, Vol 5, #2, July/August 1983, pg 22

Ken recently completed his graduate degree in Physics at the University of New Mexico. He has been programming small computers for over 15 years, and doesn't trust a computer that he can't carry, or software that doesn't come with the source code. He has worked in a variety of fields, ranging from ranching in Colorado, to Laser development at the Los Alamos National Laboratory.


```
SCR # 115
 Ø CR ." TIME FUNCTION " BASE @ DECIMAL
 1 : TI 141 C@ 256 * Ø 142 C@ Ø D+ DROP 256 U* 143 C@ Ø D+ ;
 2 : SEG Ø # # DROP DROP ; : COLON 58 HOLD ; : LOC -32728 HLD ! ;
 3 : EMITTIME LOC SEG COLON SEG COLON SEG ;
 4 : SMH  6Ø M/MOD ROT DROP 36ØØ M/ SWAP Ø 6Ø M/ SWAP ;
 5 : TIME BASE @ DECIMAL TI SMH EMITTIME BASE ! ;
 6 : CLOCK BEGIN TIME SLEEP AGAIN ;
 7 CR ." BOUNCING BALL "
 8 1 VARIABLE DX  1 VARIABLE DY
 9 : XCHECK DUP Ø= OVER 38 > OR IF DX @ MINUS DX ! ENDIF ;
1Ø : YCHECK DUP Ø= OVER 5 > OR IF DY @ MINUS DY ! ENDIF ;
11 : POS OVER OVER 4Ø * + 32768 + ; : DRAW 128 TOGGLE ;
12 : MOVE YCHECK SWAP XCHECK DX @ + SWAP DY @ + ;
13 : BB 1 3 POS DRAW
14     BEGIN MOVE POS DRAW SLEEP AGAIN DROP DROP ;
15 BASE ! -->

SCR # 116
 Ø CR ." DEFINE NEW FORTH SYSTEM INPUT STRUCTURE " BASE @ HEX
 1 CODE (KEY) XSAVE STX, FFE4 JSR, XSAVE LDX, PUSHØA JMP, FORTH
 2 : CRSON 1ØØ A7 ! ; ( TURN CURSOR ON )
 3 : CRSOFF 1 A8 C! BEGIN AA C@ UNTIL 1 A7 C! ; ( CURSOR OFF )
 4 : KEY CRSON BEGIN (KEY) SLEEP -DUP UNTIL CRSOFF ;
 5 : EXPECT ( TERMINAL INPUT MEMORY-2 )
 6    OVER + OVER DO KEY DUP 9D = IF DROP 14 ENDIF DUP 14 = ( DEL? )
 7     IF OVER I = DUP R> 2 - + >R - ELSE ( NOT DEL ) DUP ØD =
 8         IF ( RETURN ) LEAVE DROP BL Ø ELSE DUP ENDIF
 9         I C! Ø I 1+ ! ENDIF EMIT LOOP DROP ;
1Ø : MQUERY TIB @ 46 EXPECT Ø IN ! ; ( LINE INPUT )
11 : MQUIT ( RESTART, INTERPRET FROM TERMINAL )
12     Ø BLK ! [COMPILE] [ BEGIN RP! CR MQUERY INTERPRET
13         STATE @ Ø= IF ."  OK " ENDIF AGAIN ;
14 BASE ! ;S

SCR # 117
 Ø CR ." MULTI-TASKING FORTH " BASE @ DECIMAL
 1 11Ø LOAD 115 LOAD
 2 6 USER SØ    8 USER RØ
 3 : MAIN BEGIN TS3 TS1 TS3 TS4 TS3 AGAIN ;
 4 : START  ( INITIALIZE STACKS AND START MULTI-FORTH )
 5     RESET.POINTERS
 6     ( NEXT 2 LINES HOOK INTERPRETER INTO SYSTEM )
 7     1 SP + C@ SØ C! 1 RP + C@ RØ !
 8     ' MQUIT CFA ' QUIT !
 9     ' MQUIT 1 INIT.TASK
1Ø     ' BB    3 INIT.TASK
11     ' CLOCK 4 INIT.TASK
12     MAIN ; HEX
13 : STOP ' Ø CFA ' QUIT ! COLD ;
14  ( STOP WILL RETURN TO REGULAR FORTH SYSTEM )
15 BASE !    ;S
```

feature



Structure Trees in FORTH

by Michael Dougherty
Littleton, Colorado

A FORTH utility that prints the structure of a FORTH word.

Introduction

When modifying a program, it is useful to know the calling structure of that program. That is, given a FORTH word, XYZ, what is every FORTH word used to define XYZ for all calling levels down to the FORTH machine language primitives? A graphical representation of the calling structure is known as a "structure chart" (refer to Structured Design by Yourdon and Constantine). "Structure tree" is the text equivalent. In FORTH, a Structure Tree is an indented listing of each "colon definition" with all of the FORTH words used to define that word.

For example, assume that the following FORTH words have been defined:

```
: A  1  ;
: B  2 +;
: C  3  ;
: D  A B C / ;
: AB A B ;
: E  AB D AB ;
```

A Structure Chart for word E is shown in Figure 1. An equivalent Structure Tree is shown in Figure 2. (In this case, the Structure Tree shows the calling levels of only the application words.) For every colon definition, all words comprising that word's definition are printed in an indented fashion. For example, in Figure 2, word D in line #8 is at level 1. Lines 8 through 16 comprise the Structure Tree of the word D covering levels 1 through 3. After the word, /, the level is back to 1 and the Structure Tree of D is completed. As can be seen, the Structure Tree contains redundancy not found in the Structure Chart.

```
SCR # 76
  Ø ( STRUCTURE TREE UTILITY        831 BYTES )
  1
  2 ( To load the utility:          )
  3 (   76 LOAD                     )
  4 (                               )
  5 ( TREE Usage:                   )
  6 ( To list the Structure Tree    )
  7 ( of word XYZ:                  )
  8 (   TREE XYZ                    )
  9
 1Ø
 11 -->


SCR # 77
  Ø Ø  VARIABLE LEVEL                ( Current level of TREE branch)
  1 1Ø VARIABLE MAX-LEVEL            ( Maximum TREE branch level )
  2
  3 : NULL ;                         ( Dummy def to get : cfa value)
  4 ' NULL CFA @  CONSTANT COLON     ( Pointer to code of : word )
  5 ' ;S      CFA CONSTANT SEMICOLON ( Terminating CFA of : word )
  6
  7 ' ØBRANCH CFA CONSTANT 'ØBRANCH  ( Words compiling arguments )
  8 ' BRANCH  CFA CONSTANT 'BRANCH
  9 ' LIT     CFA CONSTANT 'LIT
 1Ø ' CLIT    CFA CONSTANT 'CLIT
 11 ' (LOOP)  CFA CONSTANT 'LOOP
 12 ' (+LOOP) CFA CONSTANT '+LOOP
 13 ' (.")    CFA CONSTANT '"
 14
 15 -->


SCR # 78
  Ø ( DRAW A BAR FOR THE CURRENT LEVEL )
  1
  2 : BAR                            ( length -- )
  3     -DUP IF                      ( If length > than Ø )
  4       Ø DO                       ( For length characters )
  5         ASCII  EMIT              ( Print a bar )
  6       LOOP
  7     ENDIF ;
  8
  9 -->
```

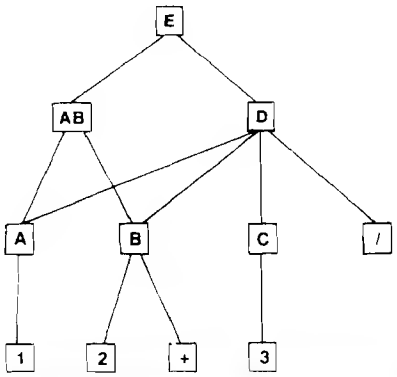


Figure 1. Sample Structure Chart.

```
E
1___AB
2______A
3_________1
2______B
3_________2
3_________+
1___D
2______A
3_________1
2______B
3_________2
3_________+
2______C
3_________3
2______/
1___AB
2______A
3_________1
2______B
3_________2
3_________+
```

Figure 2. Sample Struture Tree (for Structure Tree Utility)

```
SCR # 79
  Ø ( CASE STATEMENT BY DR. C. E. EAKER, FORTH DIMENSIONS [V2,#3]  )
  1
  2 : DOCASE  ?COMP CSP @ !CSP 4 ; IMMEDIATE
  3
  4 : <<    4 ?PAIRS COMPILE OVER COMPILE = COMPILE ØBRANCH
  5   HERE Ø , COMPILE DROP 5 ; IMMEDIATE
  6
  7 : >>       5 ?PAIRS COMPILE BRANCH HERE Ø ,
  8   SWAP 2 [COMPILE] ENDIF 4 ; IMMEDIATE
  9
 1Ø : ENDCASES 4 ?PAIRS COMPILE DROP
 11   BEGIN SP@ CSP @ = Ø= WHILE
 12     2 [COMPILE] ENDIF REPEAT
 13   CSP ! ; IMMEDIATE
 14
 15 -->

SCR # 8Ø
  Ø ( PRINT THE ARGUMENT )
  1
  2 : PRINT-ARG                              ( addr n -- addr )
  3
  4     DOCASE
  5       1 << 2 + DUP C@ .  1 +           >> ( Skip/print 1 byte )
  6       2 << 2 + DUP  @ .  2 +           >> ( Skip/print 2 bytes )
  7       3 << 2 + DUP C@ SWAP 1+ SWAP        ( Skip/print n bytes )
  8            Ø DO                          ( For len of string )
  9              DUP C@                      ( Get a dim str char )
 1Ø              EMIT                        ( Print it )
 11              1+                          ( Next string addr )
 12            LOOP                       >>
 13     ENDCASES ;
 14
 15 -->

SCR # 81
  Ø ( MOVE PFA ON STACK TO NEXT WORD )
  1
  2 : MOVE-WORD                          ( addr -- addr+offset )
  3     DUP @                            ( Get cfa compiled at ptr )
  4     DOCASE                           ( Check for special move )
  5       'ØBRANCH << 2 PRINT-ARG  >> ( Skip 2 byte offset )
  6       'BRANCH  << 2 PRINT-ARG  >> ( Skip 2 byte offset )
  7       'LIT     << 2 PRINT-ARG  >> ( Skip 2 byte value )
  8       'CLIT    << 1 PRINT-ARG  >> ( Skip 1 byte value )
  9       'LOOP    << 2 PRINT-ARG  >> ( Skip 2 byte offset )
 1Ø       '+LOOP   << 2 PRINT-ARG  >> ( Skip 2 byte offset )
 11       '"       << 3 PRINT-ARG  >> ( Skip string till " )
 12              SWAP 2 + SWAP        ( Skip only the ptr itself )
 13     ENDCASES ;
 14
 15 -->
```

The Structure Tree serves as a "roadmap" of the FORTH application. As in a roadmap, the Structure Tree does not tell how execution travels through the words. Instead, the Structure Tree provides all the words which could be called during the execution of a specific word. When modifying another programmer's code (or your own after a sufficient amount of time), knowing who calls whom will help identify the words which should be changed.

Tree

The Structure Tree Utility, TREE, is defined in Listing 1. TREE generates a Structure Tree from the application dictionary in memory. Once the TREE and application are LOADed, the Structure Tree of word XYZ is printed by:

TREE XYZ

The output may be directed to the printer by the Atari fig-FORTH word PON. Since FORTH is a highly nested language, a Structure Tree listing can be quite lengthy. (The Structure Tree of "." requires two and a half pages!) To limit the nested listing, two parameters are used in TREE. If a definition of a word is below the current value of FENCE, the Structure Tree will not continue to a lower level for that particular "tree branch." Further, the

maximum level of the Structure Tree may be limited by the variable MAX-LEVEL. In either case, a Structure Tree branch will be terminated when TREE encounters a word not defined as a colon definition (i.e., VARIABLE, CONSTANT, CODE, etc.). TREE may be aborted while listing by simply pressing any key other than BREAK.

Upon execution, TREE determines the parameter field address (pfa) of the next input word with "tick" ('). The name is printed and DO-TREE is used to print the actual Structure Tree. DO-TREE basically executes a loop until there are no parameter field addresses on the stack (i.e., when variable LEVEL goes to zero). If the parameter field address on top of the stack does not point to the end of the definition (;S), the name of that word is printed with indentation determined by LEVEL. IF the word is a colon definition not defined below FENCE, and LEVEL is less than MAX-LEVEL, then this new pfa is pushed onto the stack, LEVEL is incremented, and the loop repeated. When the end of the colon definition is reached, the pfa is popped off the stack, DO-TREE moves the pfa to the next word, LEVEL is decremented and the loop again repeated. When LEVEL goes to zero, the last pfa has been popped and DO-TREE is finished.

The only problem with the FORTH dictionary is that words such as BRANCH, LIT, and ." compile arguments directly into the word being TREEed. The word MOVE-WORD is designed to skip these compiled arguments. If your application has additional defining words which compile arguments into the definition, then MOVE-WORD will have to be extended.

Conclusion

When faced with modifying a FORTH application written by another programmer, the Structure Tree generated by TREE can be an invaluable tool. The Structure Tree allows one to determine how a word is reached during execution, as well as who the word calls.

Reference

Structured Design, Fundamentals of a Discipline of Computer Program and Systems Design by Edward Yourdon and Larry L. Constantine.

```
SCR # 82
  Ø ( PRINT THE NAME WHOSE CFA IS POINTED TO BY STACK ADDR )
  1
  2 : PRINT-NAME                    ( addr --- )
  3     CR                          ( New line for output )
  4     LEVEL @ 3 .R                ( Print the level number )
  5     LEVEL @ 3 * BAR             ( Output a bar 3*LEVEL )
  6     @                           ( Get the cfa )
  7     2 +                         ( Move to the pfa )
  8     NFA                         ( Move to nfa )
  9     ID. ;                       ( Print name )
 1Ø
 11 -->


SCR # 83
  Ø ( PRINT A WORD TREE )
  1
  2 : DO-TREE                         ( pfa --- )
  3     1 LEVEL !                     ( Init level to first )
  4     BEGIN                         ( Until the stack is empty )
  5       ?TERMINAL IF ABORT ENDIF    ( Abort TREE if key pressed )
  6       DUP @ SEMICOLON = Ø= IF     ( If not at a ;S word )
  7         DUP PRINT-NAME            ( Print the word down below )
  8         DUP @ @ COLON =           ( If lower word is a : def )
  9         LEVEL @ MAX-LEVEL @ < AND ( ...and less than MAX-LEVEL )
 1Ø         OVER @ FENCE @ > AND IF   ( ...and greater than fence )
 11           1 LEVEL +!              ( Go down to the next level )
 12           DUP @ 2 +               ( Get the pfa of that level )
 13 -->


SCR # 84
  Ø ( PRINT A WORD TREE, CONT'D )
  1
  2         ELSE                    ( Not a colon )
  3           MOVE-WORD             ( Move over to next )
  4         ENDIF
  5       ELSE                      ( End of a colon definition )
  6         -1 LEVEL +!             ( Pop up to next level )
  7         DROP                    ( Drop the addr pointer )
  8         MOVE-WORD               ( Move over )
  9       ENDIF
 1Ø     LEVEL @ Ø= UNTIL ;          ( Until stack is empty )
 11
 12 -->


SCR # 85
  Ø ( USER ENTRY FOR TREE )
  1
  2 : TREE                          ( ... TREE word ... )
  3     CR
  4     [COMPILE] '                 ( Get pfa of next input word )
  5     DUP NFA ID.                 ( Print word to be TREEed )
  6     DO-TREE                     ( Print Structure Tree of pfa )
  7     CR ;
  8
  9 ;S
```

Text Write Edit Read Program (T.W.E.R.P.)

Now reading, writing and editing textfiles is easy.

by N.D. Greene
Storrs, Connecticut

Introduction

Sequential text files provide a method for quickly storing and retrieving data. They are essential for data-based programs (such as telephone number listing, accounts receivable, etc.). Also, they have other useful applications. Programs can be shortened by storing variables and strings in a text file and loading them after the program is run (viz. during the first few lines). If formatted properly, text files may be used as exec files which greatly extend programming flexibility. For example, it is possible to rewrite a program while it is running using an exec file.

Unfortunately, text files are hard to use. There is no direct command to list their contents. You cannot "see" them except indirectly. Files may contain extra data or spaces and it is often difficult to check for these conditions. To write an exec file it is necessary to create a cumbersome exec file program.

The Textfile Write Edit Read Program (T.W.E.R.P.) was written to help write and edit text files. Using the program it is possible to read any sequential text file and to add, remove or edit any line within the file. Error trap routines detect and prevent user mistakes.

Program Description

A program listing is shown in Figure 1. The main program is only five lines long (lines 10-50). It calls a series of subprograms to perform various functions (e.g. load, read, save). These subprograms call other subroutines to run frequently used operations. The program has been written in modular form with the modules separated by remark statements. These are used only to make it easier to review the listing — they are not required to run the program.

It may be helpful to briefly review the program, since it contains several routines which might be useful in your programs. TWERP uses a command rather than a self-prompting menu. The program does not ask questions; it waits for user commands. No cursor appears unless the user has activated a command which requires further input. Nine, single, keystroke mnemomic commands (e.g., <R> READ) are used, and they remain in view at all times. A keypress is detected with the WAIT command (line 1200) and its value returned as K$. This is examined and the program is routed to the appropriate subprogram using the short routine in lines 30-50. If any key other than a code key is pressed, the control is returned to line 30.

Files are loaded from either the keyboard or disc. As prompted by the program, entering K activates the keyboard mode. Entering three *'s terminates entry. Since the length of a text file is often unknown, it is difficult to read it completely without an out of data error and program termination. This problem is avoided by using the ONERR command (line 5) and then reading a disc text file until an error occurs. Line 1000 tests for the out of data error code (5). If it is present, the file is closed and the total number of lines (fields) is calculated by subtracting 1 from the value of the loop counter, I, that caused the error.

The file in memory may be reviewed by the <R> READ command. It may be saved at anytime (with the option of changing its name). Adding or deleting lines is done by algorithms which insert or remove

lines and then renumber the file. Adding a line except at the end of the file, displaces the original lines upward. For example, if a new line number 3 is inserted, the original line number 3 becomes 4, the original line number 4 becomes 5, and so on. Similarly, if a line is deleted, all lines with greater numbers are displaced downward. If the < E> EDIT command is chosen, the current line is displayed together with a new, blank line for entering changes. This new line replaces the original.

The commands for catalog, print and quit simply list the contents of the current disc, print the file in memory or end the program. The printout should work with most printers. However, if special commands are needed, these may be entered at line 830 (printer on) and line 850 (printer off).

Error traps start at line 1000. As noted before, this line is used to detect the end of a data file. Lines, 1010, 1030 and 1040d are a universal error routine which may be used in any program. Line 1020 tests to see if any records are in memory.

The subroutines are used by the various subprograms extensively. All inputs are entered via the input anything subroutine*. Commas and other characters forbidden by the normal input command may be entered and this routine limits the length of lines to 255 characters. Backspacing erases characters - a very helpful feature. The wait routine mentioned above, waits for a keypress by checking address 49152 for a value greater than 128. It then converts this value into a normal ACSII character and stores it is K$. The wait address is then reset by a poke to 49168. This is an excellent routine for single keystroke menus. The wink cursor is a similar subroutine, frequently used by Beagle Bros.* in their commercial programs. It loops through the wait and reset addresses while overprinting to achieve a blinking effect. Center title is also a useful programming tool. The caption to be centered is sent to the subroutine as an M$ string together with the desired vtab, V. Since arrow labels are used throughout the program, it was written here as a subroutine.

Program Applications

This program works with Apple II, II+ and II/e Computers**. Errors may occur if it is run on systems with DOS moved to a language card. Using TWERP, text file manipulation is almost as easy as programming in BASIC. It is especially easy to create exec files, one of the more powerful and less used routines available in Applesoft**. TWERP also provides an easy way to rename text files and/or to transfer them to another disk.

Acknowledgements

Thanks to R.H. Gandhi for his helpful suggestions and to L. Fosdick for supplying the input anything routine and his permission to use it in this program.

*Written by L. Fosdick, E. G. & G. Princeton Applied Research, Princeton, NJ

*Beagle Bros, Inc., San Diego, CA

**Registered Trademark

Listing 1

```
          ============================
          TEXTFILE WRITE EDIT READ PROGRAM

                    (T.W.E.R.P)

                    N.D. GREENE
                 COPYRIGHT (C) 1983
          ================================
    5  D$ =  CHR$ (13) +  CHR$ (4): DIM S$(500):
       POKE 44452,19: POKE 44605,18:
       ONERR GOTO 1000
    6  REM
    7  REM      MAIN PROGRAM
    8  REM
   10  HOME :V = 10:
       M$ = "TEXTFILE WRITE EDIT READ PROGRAM":
       GOSUB 1190:V = 12 : M$ = "(T.W.E.R.P.)":
       GOSUB 1190: PRINT : VTAB 21:
       FOR I = 1 TO 40: PRINT
        "=";: NEXT
   20  VTAB 22:
       PRINT "< C> CATALOG" TAB( 17)"< A> ADD" TAB( 32)
       "< P> PRINT";:
       PRINT < L> LOAD" TAB( 17)"< D> DELETE" TAB( 32)
       "< S> SAVE ";: PRINT "< R>
       READ" TAB( 17)" < E> EDIT" TAB( 32)"< Q> QUIT";
       :VTAB 4: POKE 35,20

   30  CALL 54915: GOSUB 1200: FOR I = 1 TO 9:
       IF K$ =  MID$ ("LRSADECPQ",I,1) THEN 50
   40  NEXT
   50  ON I GOSUB 100,200,300,400,500,600,700,800,
       900,30
   60  REM
   61  REM      LOAD FILE
   62  REM
  100  HOME : VTAB 10: HTAB 5: PRINT "FILE NAME:";
  110  V = 20:M$(1) = " D  DISK   < K> KEYBOARD":
       M$(2) = "< D> DISK   < K>  KEYBOARD": GOSUB 1210:
       IF K$ <  > "D" AND K$ <  > "K" THEN 110
  120  VTAB 20: HTAB 1: CALL  - 868:
       IF K$ = "K" THEN F$ = "KEYBOARD":ST = 1:
       GOTO 170
  130  VTAB 10: HTAB 16: GOSUB 1100
  140  F$ = T$: PRINT D$;"VERIFY";F$:
       PRINT D$;"OPEN";F$:: PRINT D$;"READ";F$:
       HOME:V = 10:M$ = "LOADING FILE.": GOSUB 1190:
       FOR I = 1 TO 500:S$(I) = ""
  150  GET K$:
       IF K$ <  > CHR$ (13) THEN S$(I) = S$(I) + K$:
       GOTO 150
```

(continued)

```
160  PRINT "..";: NEXT : GOTO 30
170  HOME :V = 1:M$ = "ENTER => *** <= TO STOP":
      GOSUB 1190: PRINT : PRINT :
      FOR I = ST TO 500: PRINT
180  NX = I: GOSUB 1230: GOSUB 1100:S$(I) = T$:
     IF S$(I) = "***" THEN N = I - 1:V = 10:
     M$ = "ENTRY ENDED AT LINE NO: " +  STR$ (N):
     HOME : GOSUB 1190:
GOTO 30
190  NEXT
191  REM
192  REM       READ FILE
193  REM
200  GOSUB 1020: HOME : FOR I = 1 TO N:NX = I:
     GOSUB 1230: PRINT S$(I): IF
      PEEK (37) < 17 THEN 220
210  V = 20:M$(1) = "PRESS ANY  KEY  FOR MORE":
     M$(2) = "PRESS ANY <KEY>  FOR
      MORE: GOSUB 1210: HOME
220  NEXT : GOTO 30
221  REM
222  REM         SAVE FILE
223  REM
300  GOSUB 1020: HOME : VTAB 10: HTAB 5:
     PRINT "FILE NAME: ";F$
310  V = 20:
     M$(1) = "< S>  SAVE   C  CHANGE  <E>  EXIT":
     M$(2) = " S  SAVE  <C>
      CHANGE   E  EXIT": GOSUB 1210:
     FOR I = 1 TO 3: IF K$ =  MID$
      ("SCE",I,1) THEN 330
320  NEXT
330  ON I GOTO 340,370,380,310
340  V = 10:M$ = " SAVING FILE "
350  HOME : GOSUB 1190: PRINT D$"OPEN"F$:
     PRINT D$"DELETE"F$: PRINT
      D$"OPEN"F$:PRINT D$"WRITE"F$
360  FOR I = 1 TO N: PRINT S$(I): NEXT :
     PRINT "CLOSE": HOME :V = 10:M$ =
      "FILE SAVED": GOSUB 1190: GOTO 30
370  VTAB 14: HTAB 5: PRINT "NEW NAME: ";:
     GOSUB 1100:F$ = T$: GOTO 300
380  HOME : GOTO 30
381  REM
382  REM      ADD LINES
383  REM
400  GOSUB 1020: HOME : VTAB 10: HTAB 5:
     PRINT "ADD LINE NO: ";: GOSUB 1100:
     NA = VAL (T$): IF NA <  1 OR NA >  N + 1
     THEN 400
410  IF NA = N + 1 THEN ST = N + 1: GOTO 170
420  HOME : VTAB 10:NX = NA: GOSUB 1230:
     GOSUB 1100
430  V = 20:M$(1) = "< A>  ADD   E  EXIT":
     M$(2) = " A  ADD  <E>  EXIT": GOSUB
     1210:I = N + 1: IF K$ = "A" THEN 460
440  IF K$ = "E" THEN 380
450  GOTO 430
460  S$(I) = S$(I - 1):I = I - 1:
     IF I = NA GOTO 480
470  GOTO 460
480  S$(I) = T$:N = N + 1: HOME :V = 10:
     M$ = "ADDED LINE NO: " +  STR$ (NA):
     GOSUB 1190: GOTO 30
481  REM
482  REM       DELETE LINES
483  REM
500  GOSUB 1020: HOME : VTAB 10: HTAB 5:
     PRINT "DELETE LINE NO: ";: GOSUB
     1100:ND =  VAL (T$): IF ND <  1 OR ND >  N THEN 500
510  HOME : VTAB 10:NX = ND: GOSUB 1230:
     PRINT S$(ND)
520  V = 20:M$(1) = " D  DELETE  <E>  EXIT":
     M$(2) = "< D>  DELETE   E  EXIT":
     GOSUB 1210: IF K$ = "D" THEN 550
530  IF K$ = "E" THEN 380
540  GOTO 520
550  IF ND = N THEN N = N - 1: GOTO 570
560  FOR I = ND TO N - 1:S$(I) = S$(I + 1):
     NEXT :N = N - 1
570  HOME :V = 10:M$ = "DELETED LINE NO: ":
     GOSUB 1190: PRINT ND: GOTO 30
571  REM
572  REM      EDIT LINES
573  REM
600  GOSUB 1020: HOME : VTAB 10: HTAB 5:
     PRINT "EDIT LINE NO: ";: GOSUB 1100:
     NE =  VAL (T$): IF NE <  1 OR NE >  N THEN 600
610  HOME : VTAB 8:NX = NE: GOSUB 1230:
     PRINT S$(NE): PRINT : PRINT : GOSUB
     1230: GOSUB 1100
620  V = 20:M$(1) = "< A>  ADD   E  EXIT":
     M$(2) = " A  ADD  <E>  EXIT": GOSUB
     1210:IF K$ = "A" THEN 650
630  IF K$ = "E" THEN 380
640  GOTO 620
650  S$(NE) = T$: HOME :V = 10:
     M$ = "EDITED LINE NO: " +  STR$ (NE):
     GOSUB 1190:  GOTO 30
651  REM
652  REM      CATALOG
653  REM
700  HOME : PRINT D$;"CATALOG":V = 20:
     M$(1) = "PRESS ANY <KEY>  TO CONTINUE
     :M$(2) = "PRESS ANY  KEY  TO CONTINUE":
     GOSUB 1210: GOTO 380
704  REM
705  REM      PRINT
706  REM
800  GOSUB 1020: HOME :V = 10:
     M$ = "TURN PRINTER ON": GOSUB 1190:V = 13:
     M$(1) = "< S>  START   E  EXIT":
     M$(2) = " S  START  <E>  EXIT": GOSUB 1210:
     IF K$ = "S" THEN 830
810  IF K$ = "E" THEN 380
820  GOTO 800
830  PR# 1: PRINT
840  FOR I = 1 TO N:
     PRINT  CHR$ (91);I; CHR$ (93);"=>  ";S$(I): NEXT
850  PR# 0
860  V = 12:M$ = "PRINTOUT COMPLETED": HOME :
     GOSUB 1190: GOTO 30
```

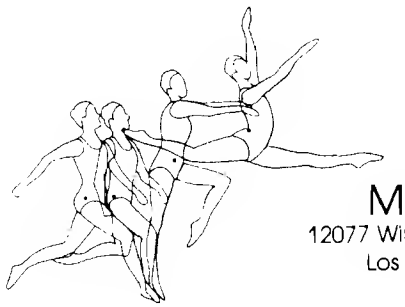
(Continued on next page)

MicroMotion

MasterFORTH

It's here — the next generation of MicroMotion Forth.

- Meets all provisions, extensions and experimental proposals of the FORTH-83 International Standard.
- Uses the host operating system file structure (APPLE DOS 3.3 & CP/M 2.x).
- Built-in micro-assembler with numeric local labels.
- A full screen editor is provided which includes 16 x 64 format, can push & pop more than one line, user definable controls, upper/lower case keyboard entry, A COPY utility moves screens within & between lines, line stack, redefinable control keys, and search & replace commands.
- Includes all file primitives described in Kernigan and Plauger's Software Tools.
- The input and output streams are fully redirectable.
- The editor, assembler and screen copy utilities are provided as relocatable object modules. They are brought into the dictionary on demand and may be released with a single command.
- Many key nucleus commands are vectored. Error handling, number parsing, keyboard translation and so on can be redefined as needed by user programs. They are automatically returned to their previous definitions when the program is forgotten.
- The string-handling package is the finest and most complete available.
- A listing of the nucleus is provided as part of the documentation
- The language implementation exactly matches the one described in FORTH TOOLS, by Anderson & Tracy. This 200 page tutorial and reference manual is included with MasterFORTH
- Floating Point & HIRES options available.
- Available for APPLE II/II+/IIe & CP/M 2.x users.
- MasterFORTH – $100.00. FP & HIRES –$40.00 each
- Publications
  - FORTH TOOLS – $20.00
  - 83 International Standard – $15.00
  - FORTH-83 Source Listing 6502, 8080, 8086 – $20.00 each.



Contact:

MicroMotion
12077 Wilshire Blvd., Ste. 506
Los Angeles, CA 90025
(213) 821-4340

(continued)

```
 862  REM
 863  REM     QUIT
 864  REM
 900  TEXT : POKE 44452,22: POKE 44605,21: HOME :
       POKE 216,0: END
 904  REM
 905  REM     ERROR TRAPS
 906  REM
1000  IF  PEEK (222) = 5 THEN  PRINT D$;"CLOSE":
      N = I - 1: HOME : VTAB 10: HTAB 12:
      PRINT N;" RECORDS LOADED": GOTO 30
1010  HOME :V = 12:M$ = "ERROR CODE:": GOSUB 1190:
      PRINT  PEEK (222): GOSUB 1040: GOTO 380
1020  IF N = 0 THEN  HOME :V = 12:
      M$ = "NO RECORDS IN MEMORY": GOSUB 1190:
      GOSUB 1040: GOTO 380
1030  RETURN
1040  FLASH :V = 10:M$ = "OPERATOR ALERT":
     GOSUB 1190: NORMAL :V = 20:
     M$(1) = "PRESS ANY <KEY> TO RESET":
     M$(2) = "PRESS ANY  KEY  TO RESET":
     GOSUB 1210: RETURN
1050  REM
1051  REM    SUBROUTINES
1052  REM
1054  REM    INPUT ANYTHING
1055  REM
1100  T$ = "":K = 0
1110  GET K$:J =  ASC (K$): IF J = 8 THEN 1150
1120  IF J = 13 THEN  RETURN
1130  IF K = 255 THEN 1110
1140  PRINT K$;:T$ = T$ + K$:K = K + 1: GOTO 1110
1150  IF K = 0 THEN 1110
1160  PRINT K$;" ";K$;:K = K - 1: IF K = 0 THEN 1100
1170  T$ =  LEFT$ (T$,K): GOTO 1110
1180  RETURN : REM
1182  REM
1183  REM   CENTER TITLE
1184  REM
1190  VTAB V: HTAB 21 -  LEN (M$) / 2: PRINT M$;:
      RETURN
1192  REM
1193  REM     WAIT FOR KEYPRESS
1194  REM
1200  WAIT 49152,128:
      K$ =  CHR$ ( PEEK (49152) - 128): POKE 49168,0:
      RETURN
1201  REM
1202  REM     WINK CURSOR
1204  REM
1210  SPEED= 220:M$ = M$(1): GOSUB 1190:M$ = M$(2):
      GOSUB 1190:K =  PEEK (49152):
      IF K < 128 THEN 1210
1220  K$ =  CHR$ (K - 128): SPEED= 255: POKE 49168,0:
      RETURN
1221  REM
1222  REM   ARROW LABEL
1223  REM
1230  PRINT : HTAB 1: INVERSE : PRINT NX;: NORMAL :
      PRINT "=>  ";: RETURN
```

MICRO

feature

Graphic Print for C-64
Part 3

by Michael J. Keryan
Tallmadge, Ohio

In the first two parts of this series, we developed a program to produce a graphic screen dump for most popular non-Commodore printers. The fast machine language program can be used to print graphic files from a number of popular graphic programs. In this last installment, we will show how to get full-color printouts using your existing dot-matrix printer.

Add full color to your graphic printouts without a color printer.

Introduction

The graphic print program described in the last two issues will give a HiRes or MULTIcolor graphic dump in various dot patterns — the density of the dot patterns is proportional to the darkness of the actual colors used in the picture. Sixteen different patterns are used so that even two colors that look identical on a black and white monitor can be distinguished on the printout. What can be better than this? Color. There are several methods that you can use to get full-color hard copies of graphic displays created on your Commodore 64.

One method is to use a good quality 35mm camera with color film and shoot the pictures displayed on your color monitor. This is probably the best way to create slides for presentations. The third party graphic programs available for the Commodore 64 will create outstanding title slides, graphs, bar charts, and pictures. This method can also be used to create larger color prints, but this can be expensive for a full-page enlargement.

A second method is to buy a color printer or plotter. Several new color dot-matrix printers have recently been introduced; some use multi-color ribbons, some use ink-jet technology. You can get one for $600-$1000+. If you want to create a lot of color prints, you can probably justify one. If you don't yet have a printer and are thinking of getting a color dot-matrix printer for general-purpose use, make sure it can produce a good sharp black letter. Many of these color printers produce only 3 colors, and create black by mixing all 3; black letters may appear as smeared gray. Color plotters are used mainly for very high resolution line plots, graphs, and charts. They are not generally used for dumping color pictures, but software could be written to do this by drawing a large number of short lines. However, this would be extremely slow.

A third method (and the one described in this article) is to use color ribbons with your existing dot-matrix printer. If you have a good quality graphic printer, want full-page printouts in color only occasionally, and are willing to invest some of your time but not much money, this method is for you. It works by overprinting the same printout several times, once for each ribbon color.

Color Ribbons

The first thing you will need is a set of color ribbons. A number of printer supply firms sell ribbons in various colors for most popular printers at about 30% higher cost than the standard ribbons. The ribbon cartridges are exactly the same as your black ribbons; the only difference is the color of the ink on the ribbon. Note that a set of ribbons is used — one for each color — multicolor ribbons are not used. Another way to obtain a set of color ribbons is to ink your own. You can buy new, uninked ribbons, place them in your old cartridges, and use one of the new mechanical inking gadgets that wind the ribbon while applying ink.

How many colors will you need? Well that depends on the type of pictures you want printed and on the colors that you can obtain. You certainly do not need all 16 colors that the Commodore 64 can produce. Many of these can be created from combinations and varying patterns of other colors. If you can find them, use black, blue, red, and yellow. All the other colors can be generated from these. Unfortunately, I could not find a source for yellow ribbons (if you know where I can get a yellow Prowriter ribbon, let me know), so I used black, blue, red, green, and brown. With

these, all colors except yellow and orange can be generated.

Keep the ribbon cartridges in plastic bags that can be sealed to keep the ink from drying out. Make sure the rollers in the cartridges turn smoothly. Nothing is more frustrating than having a ribbon stick halfway through the last color of a five color printout.

Helpful Hints

As previously noted, the color prints are made by printing over the same page (or pages) several times, once for each color. A lot of things can go wrong while you're doing this. Here are a few pointers that I've picked up in the last several months; they may save you some time and grief:

1. Be prepared to spend an hour or more to get a few prints. If you are only printing one, make several copies — 3 or 4 to be safe. That way if a ribbon jams or a page is misaligned, the others should still be good. Save up your pictures to be printed and do them all in one session. This will reduce the number of ribbon swaps.
2. If you are using pin-feed paper, prepare the paper beforehand. The perfs tend to tear apart when moving the paper up and down several times to get all colors printed. A partially torn perf will catch and cause the paper to jam. Prior to printing, take out the paper, tape over the perfs with scotch tape, and then place it back in your printer. You can even form a continuous loop with several sheets, avoiding the need to backtrack the platen.
3. Prior to printing the first color on each page, draw an index line in pencil, aligning the mark to some stationary point on the printer carriage or frame. For subsequent colors, make sure this alignment mark is in the same place.
4. A misalignment of even one dot (1/72 inch) is quite noticeable. If a misalignment is obvious while printing a page, turn off the printer with the off-line or select switch on the printer, then try to align the page by adjusting the platen. Although you won't be able to save the current page, subsequent pages (see tip 1) should then end up perfectly aligned.
5. Some printers have a panel that must be removed when changing ribbons. A micro switch will not allow printing when this panel is removed. To save you some time, remove this panel and tape down the micro switch to defeat it. Then leave the panel off while you are getting your color prints.

Even by using all the above precautions, I've yet to get correct alignment of all colors on the first page of 4 copies that I print. My average is about 50% of all copies with no noticeable misalignment. But the results are rewarding.

Color Print

A BASIC program to create color printouts is given in Listing 1. This program is based on the program in last month's MICRO. All the machine language routines are the same. The only changes necessary are the printer matrix codes for each color. These codes change with each ribbon color. The program first loads, from diskette, the machine language routines "GDUMP+MOVE", then jumps to line 2000. Here the type of picture is selected. Then in lines 2120-2250 the printer specific information is defined. Change the values for variables PT, NT, and SD for your printer/interface set-up, then save the revised program. The picture file is then loaded into memory from disk. While loading this file, the CRT will display the following menu:

```
AFTER PICTURE LOADS, PRESS:
P FOR BLACK/WHITE PRINT

COLOR PRINT:
   B FOR BLACK RIBBON
   N FOR BROWN RIBBON
   U FOR BLUE RIBBON
   R FOR RED RIBBON
   G FOR GREEN RIBBON

   E TO EXIT
```

The program works as last months program for black/white printouts when pressing P. For color prints, first set up the paper and the first ribbon, then press the key that corresponds to that color. After one (or more) prints of that particular color are finished, change the ribbon, set up the paper back to the beginning, then press the key for the new color. Keep this up until all required colors are printed. Then press E to exit.

The color pattern information in matrix CM (see lines 2350-2360) is POKEd to the machine language routine in lines 1150-1320, then the printer dump is executed (line 1330). This loop is repeated for each color.

Dots are printed for each combination of screen color and ribbon color containing an X in the table below. A high density of dots is printed for dark colors, a lower density for lighter colors.

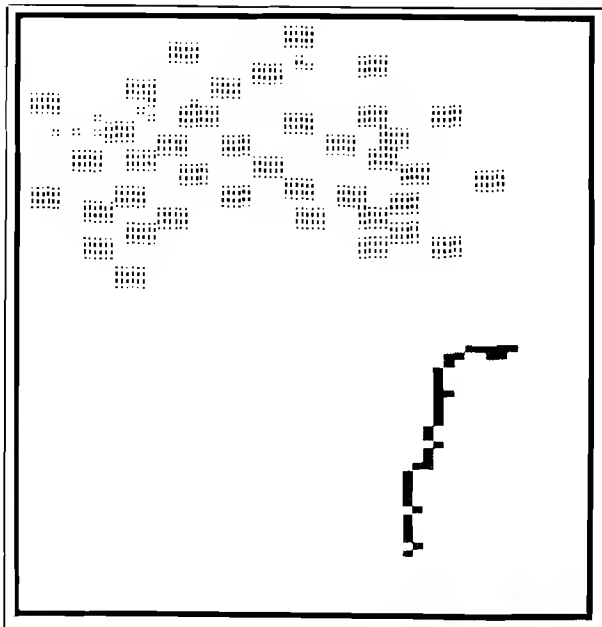
Screen Color	Ribbon Color: Black	Brown	Red	Green	Blue
Black	X	-	-	-	-
White	-	-	-	-	-
Red	-	-	X	-	-
Cyan	-	-	-	X	X
Purple	-	-	X	-	X
Green	-	-	-	X	-
Blue	-	-	-	-	X
Yellow	-	X	-	-	-
Orange	-	X	X	X	-
Brown	-	X	-	-	-
Light Red	-	-	X	-	-
Gray 1	X	-	-	-	-
Gray 2	X	-	-	-	-
Light Green	-	-	-	X	-
Light Blue	-	-	-	-	X
Gray 3	X	-	-	-	-

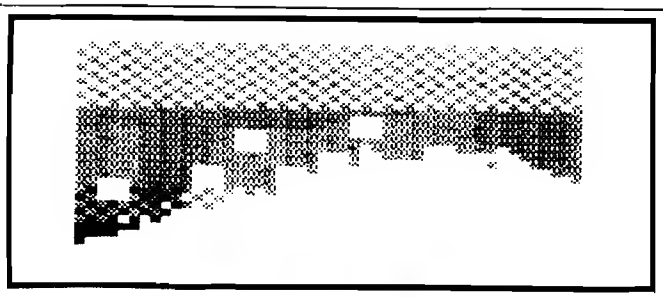
If you have ribbons of different colors or would like to experiment with different shades or color codes, change the data in lines 4000-4050. Line 4000 contains the pattern codes as described in part 1 of this series. The other DATA lines (one line for each ribbon) point to the 0 to 15th term in line 4000. There are 16 entries in each line, one entry for each color (0 black, 1 white, etc.). All zeroes in these lines will print no dots for those color/ribbon combinations.

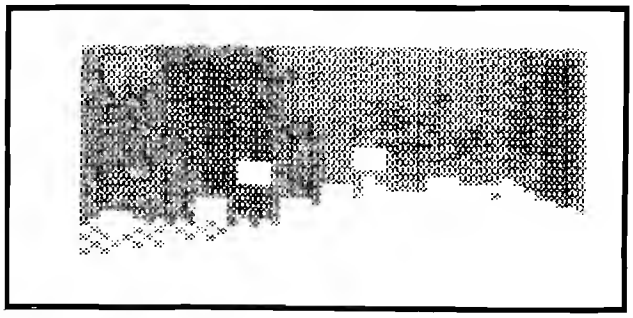
The programs described in this 3 part series of articles can be obtained on 1541 format disks for $15 (US) from MICRO (disk number MD-4).

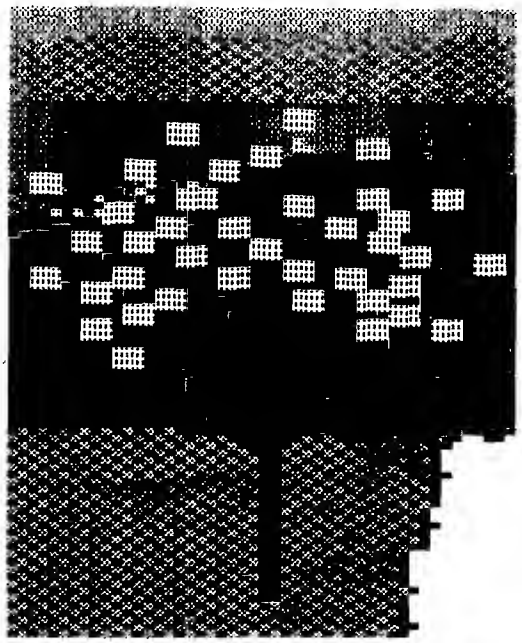
[Editors Note: The picture on page 22 of the July issue (part 2 of this series) is titled "MIDDLE EARTH" and was created by Wayne Schmidt of New York City, using **Doodle** *by City Software and is a demo on the Doodle disk. Credit was inadvertantly omitted.]*

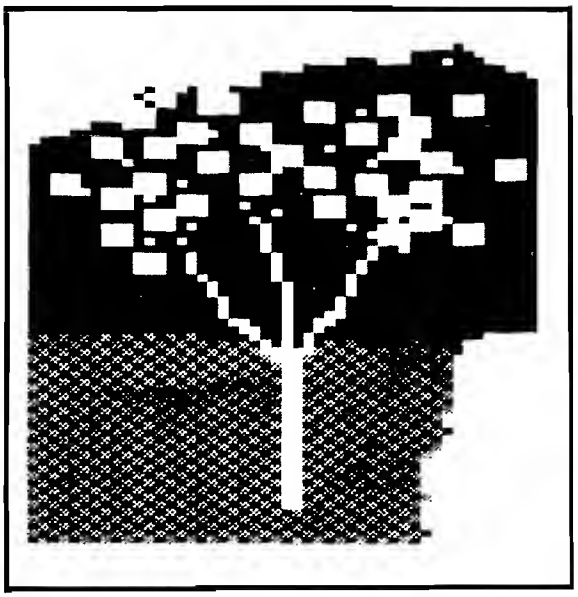
Michael J. Keryan has written a number of articles in *MICRO*, *BYTE*, and *COMPUTER and ELECTRONICS* involving machine language utility programs and hardware add-ons for various microcomputers, including OSI, TRS-80 and Commodore 64. His *DOES-IT* program to extend the capabilities of the C64 by implementing keyboard-callable machine language routines and supporting swapping of machine language programs appeared in recent issues of *MICRO* (January through April/May 1984).

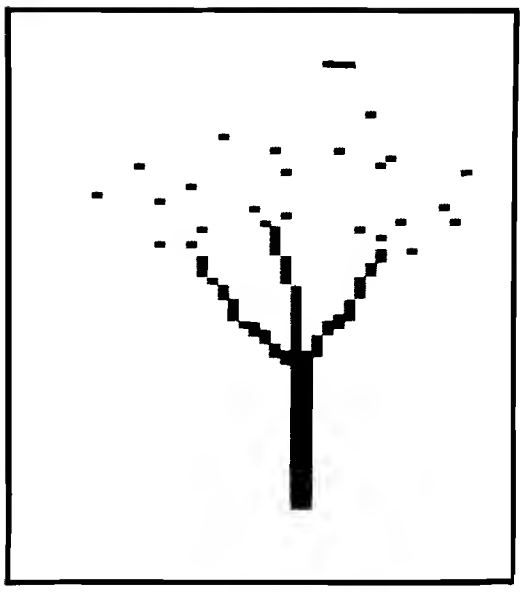












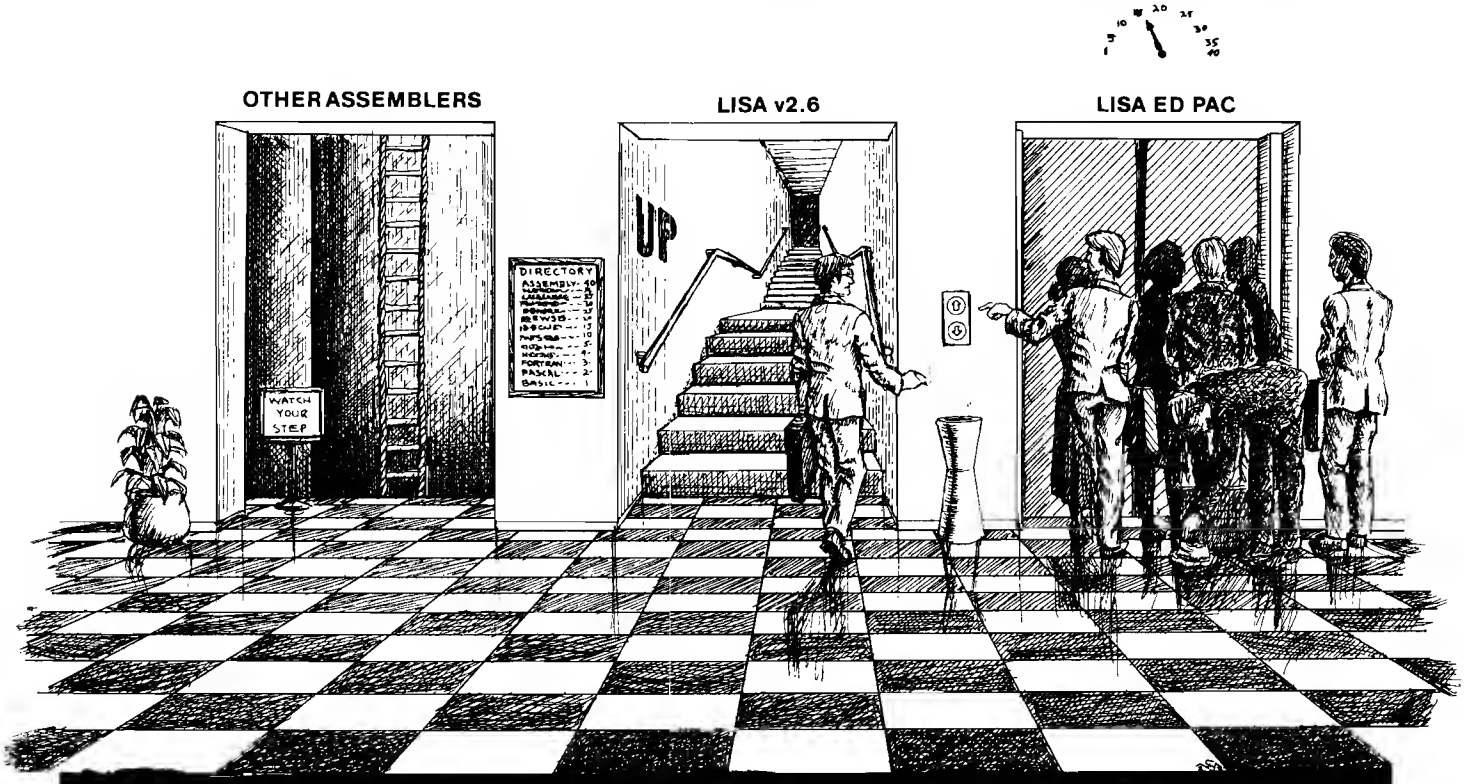
Listing 1

```
1000 REM BASIC PROGRAM TO SUPPORT GDUMP
1005 REM PROVIDES COLOR PRINTOUTS
1010 REM     M.J.KERYAN  3-30-84
1020 :
1030 IF A=0 THEN A=1: LOAD" GDUMP+MOVE",8,1
1040 IF A=1 THEN A=2: GOTO 2000
1050 POKE 20491,PT: POKE 20492,SD
1060 POKE 20493,NT: POKE 20487,NP
1070 SYS GT
1080 IF TY=2 OR TY=4 THEN MD=PEEK(53270):
     MD=3-((MD AND 16)/16): POKE 20494,MD
1090 IF TY=3 OR TY=5 THEN POKE 20494,3
1100 IF TY=6 THEN POKE 20494,0
1110 GETK$:IF K$<>""THEN 1110
1120 GETK$:IF K$="" THEN 1120
1130 IF K$="P" THEN SYS 20480: GOTO 1800
1140 IF K$="E" THEN 1800
1150 C=0: IF K$="B" THEN C=1
1160 IF K$="N" THEN C=2
1170 IF K$="R" THEN C=3
1180 IF K$="G" THEN C=4
1190 IF K$="U" THEN C=5
1200 IF C=0 THEN 1110
1300 FOR M=0 TO 15: MM=21182+M: NN=21198+M
1310 POKE MM,CM(C,M): POKE NN,CM(0,M)
1320 NEXT M
1330 SYS 20480: GOTO 1110
1800 :    REM QUIT
1840 POKE 53265,(PEEK(53265)AND223)
1850 POKE 53270,(PEEK(53270)AND207)
1860 POKE 53272,21
1870 POKE 53280,6: POKE 53281,15: POKE 646,0
1880 PRINT"{CLEAR}": END
2000 POKE 53280,6: POKE 53281,15: POKE 646,0
2010 PRINT"{CLEAR,DOWN2}WHICH TYPE OF PICTURE?"
2020 PRINT
2030 PRINT"    1  SIMON'S BASIC"
2040 PRINT"    2  ULTRABASIC-64"
2050 PRINT"    3  DOODLE"
2060 PRINT"    4  KOALAPAINTER"
2070 PRINT"    5  SLIDESHOW"
2080 PRINT"    6  SLIDESHOW - INVERTED"
2090 INPUT"  ";TY
2100 IF TY<1 OR TY>6 THEN 2000
2110 :
2120 PT = 0:  REM PRINTER TYPE
2130 :        REM  NEC/PROWRITER    = 0
2140 :        REM  EPSON OR SIMILAR = 1
2150 :
2160 NP = 3:  IF PT=1 THEN NP = 2
2170 :        REM  REPEAT CODE
2180 :
2190 NT = 0:  REM INTERFACE TYPE
2200 :        REM  CONNECTION = 0
2210 :        REM  OTHERS     = 1
2220 :
2230 SD = 6:  REM SECONDARY ADDRESS
2240 :        REM FOR TRANSPARENT
2250 :
2260 GT = 21808 + (TY-1)*3
2270 IF GT>21820 THEN GT=21820
2280 IF TY=1 THEN 3000
2290 PRINT"{DOWN2}NOW PUT IN DISK WITH THE PICTURE FILE."
2300 INPUT"{DOWN}NAME OF PICTURE";NM$
2310 PRINT"{DOWN}AFTER PICTURE LOADS, PRESS:"
2320 PRINT"   P FOR BLACK/WHITE PRINT"
2321 PRINT
2322 PRINT"   COLOR PRINT:"
2323 PRINT"   B FOR BLACK RIBBON"
2324 PRINT"   N FOR BROWN RIBBON"
2325 PRINT"   U FOR BLUE RIBBON"
2326 PRINT"   R FOR RED RIBBON"
2327 PRINT"   G FOR GREEN RIBBON"
2329 PRINT
2330 PRINT"   E TO EXIT"
2340 DIM CM(5,15)
2350 FOR I=0 TO 5: FOR J=0 TO 15
2360 READ MM: CM(I,J)=MM: NEXTJ: NEXTI
2370 IF TY=4  THEN LOAD "?"+NM$+"*",8,1
2380 IF TY<>4 THEN LOAD NM$+"*",8,1
2900 :
3000 REM CREATE A SIMON'S BASIC PROGRAM
3010 Q$=CHR$(34)
3020 PRINT"{CLEAR}1 IF A=1 THEN A=2:
     LOAD"Q$" GDUMP+MOVE"Q$",8,1"
3030 PRINT"2 IF A=0 THEN A=1: GOTO 7
3040 PRINT"3 POKE 20491,"PT" :POKE 20492,"SD
3050 PRINT"4 POKE 20493,"NT" :POKE 20487,"NP":
     SYS 21808"
3060 PRINT"5 A=PEEK(53270): A=(A AND 16)/16"
3070 PRINT"6 A=3-A: POKE 20494,A: SYS 20480: END"
3080 PRINT"7 REM APPEND YOUR PROGRAM HERE"
3090 PRINT"SAVE"Q$"SIMON.GDUMP"Q$",8"
3100 POKE 631,19: FOR A=632 TO 639: POKE A,13: NEXT A
3110 POKE 198,9: NEW
4000 DATA 0,5,32,10,64,20,1,40,159,165,90,
         130,219,135,80,255
4010 DATA 15,0,0,0,0,0,0,0,0,0,0,12,13,0,0,3
4020 DATA 0,0,0,0,0,0,0,0,2,4,15,0,0,0,0,0,0
4030 DATA 0,0,15,0,9,0,0,0,6,0,9,0,0,0,0,0,0
4040 DATA 0,0,0,5,0,15,0,0,2,0,0,0,0,9,0,0
4050 DATA 0,0,0,11,10,0,15,0,0,0,0,0,0,0,10,0
```

MICRO

There are three ways to learn 6502 Assembly Language on your Apple Computer:

Hard Easy Easiest



Introducing the Easiest Way: The LISA Ed Pac™

You can't deny that learning assembly is extremely important for you if you want to make the most of your work. If assembly language wasn't so important, why are almost all of the top selling programs available for the Apple II written in assembly language? But let's face it, learning 6502 assembly language isn't a piece of cake. At least not until now. Because now there's the LISA Education Package™ from Lazerware. It'll have you up to speed with assembly language in a fraction of the time it would otherwise take.

The LISA Ed Pac™ begins with LISA v2.6, the favorite assembler of beginners and professionals alike. More Apple owners have learned 6502 assembly language using LISA than all the other assemblers combined. More tutorial material is available for LISA, including books by D. Fudge, R. Hyde, W. Maurer, and R. Mottola. Randy Hyde's 300-page *Using 6502 Assembly Language* is included in the LISA Ed Pac™.

Next we threw in SPEED/ASM™, a set of 6502 subroutines that make programing in assembly language as easy as BASIC. And for those who want to see how it's done, the SPEED/ASM source listings are also included. We also included the LUD # 1 (Lisa Utility Disk # 1) which includes an extended editor for LISA and a LISA source file listing utility. Finally, we added MAXWELL'S Debugger™ to the LISA Ed Pac. This ultra-powerful debugger/monitor makes learning and debugging 6502 assembly language a breeze.

LISA Ed Pac Price $149.95, A $229.75 Value (suggested retail)
Available at dealers everywhere, or directly from

LAZERWARE

For a copy of Lazwerware's *A Guide to Purchasing a 6502 Assembler for Your Apple II or Apple IIe* write to: Lazerware, [illegible] Corona, CA 91720 or call us at (714) 735-1041
Note: LISA, LISA v2.6, LISA Ed Pac, LISA Educational Package, Speed/ASM, Maxwell's Demo [illegible]
Apple, Apple II, and Apple IIe are trademarks of Apple Computer Inc.

Approximating the Square Root of the Sum of the Squares

A fast method of calculating this useful function.

by Chris Williams
Ogden, Utah

If I were asked to make a list of what I thought were the most often executed computer calculations in the world, very near the top of that list would be the following equation:

```
S=SQR((X**2.)+5(Y**2.))
```

This square root of the sum of the squares calculation is used for all kinds of different things. It crops up in electronics, in physics, in geometry, and in just about every other imaginable field. It is widely used because it is the equation for calculating the magnitude of two rectangular components — of anything. It tends to be a repetitive calculation, especially in the world of graphics and animation where computers are called on to perform it hundreds of times.

Since this is true, a computer technique that will save just one microsecond executing the calculation is precious. Using such a technique in a repetitive application subtracts one microsecond from the execution time for each loop and you wind up finishing your task much faster than you otherwise would have.

Bearing all that in mind, consider the value of the following technique; it saves tens or hundreds of microseconds for each repetition.

Skeptical? Don't be. The method is incredibly simple. It is incredibly fast. It is, without further ado:

```
SPRIME=a*X + b*Y
```

That's it.

It's an approximation, of course, but please, keep reading. With the proper choice of a and b the peak error of this approximation is a grand total of 4.01 percent. No misprint there, a maximum error of 4.01 percent.

Ah, good. You're paying attention again. Now then, think of how much faster your computer will be able to do SPRIME = a*X + b*Y than it could S = SQR((X**2.) + (Y**2.)).

The optimal (from a minimal peak error perspective) values of a and b are a = 0.961 and b = 0.398, to three decimal places. Use those and SPRIME will never vary from S by more than 4.01 percent. For those of you who haven't realized it yet, you probably will never see graphics errors that small.

For the assembly language inclined readers, here's another tidbit of value. Suppose we choose values of a and b that are related to powers of two. The a*X and b*Y operations then become simple shifts of X or Y an amount equal to what the power of two a or b is. Doing that is thousands of times faster than a full-fledged floating-point multiply.

It turns out that the optimal binary-related values are not exactly powers of two. The values are a = 1. (no problem there) and b = .375, again to three decimal places. Since b is not an exact power of two you have to do a little more than just a simple shift. Namely, shift Y three times to the right (.125*Y) and add the result to itself twice (.125*3 = .375). That's still much faster than any floating-point multiply.

Incidently, this combination of coefficients yields a peak error of 6.8 percent.

Is there a catch? Yes, but it's a small one. Your X must be greater than Y. If it isn't to start with, simply switch coefficients.

A similar procedure can be done in 3-D. In that case, we'd be approximating R = SQR((X**2.) + (Y**2.) + (Z**2.))), and we'd do so with RPRIME = ((a*X) + (b*Y) + (c*Z))). The optimal, binary-related coefficients are a = 1.0, b = .375 and c = .25. The peak error is 9.68 percent. This is allowing a maximum right shift of three. You could achieve superior peak error performance if you shifted more, but you'd lose significance as bits were shifted off the end of the byte. Three seems a good compromise.

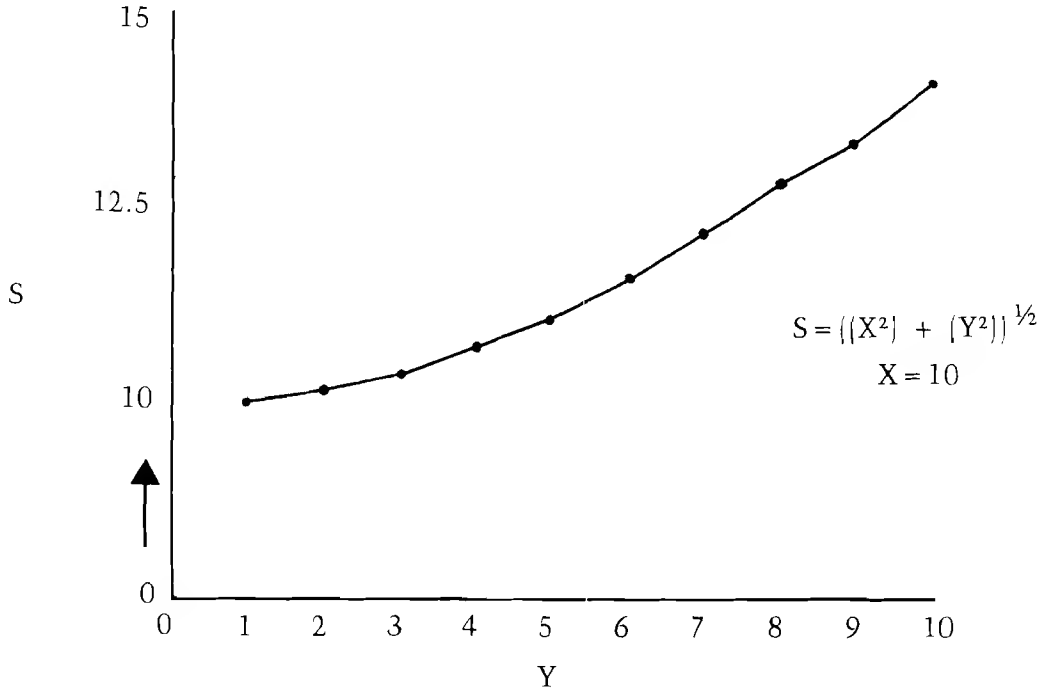
Accompanying this article is a program that demonstrates the validity of the above claims. It's written in Applesoft, but there's nothing particularly machine dependent in it, so you should be able to get it to run on any BASIC machine.

The outputs are shown in Fig.2. The S value represents the results from Applesoft's straightforward SQR((X**2.) + (Y**2.)) calculation. SPRIME is the approximation value. Both are computed for $0 < X/Y < 1.0$ in steps of 0.1. Error is computed as a percentage and is ((S - SPRIME) / S) * 100.

So that's it. It's a good technique. Try it.

Reference

Magnitude Approximations for Microprocessor Implementation by W. Thomas Adams and John Brady, IEEE Micro, October 1983



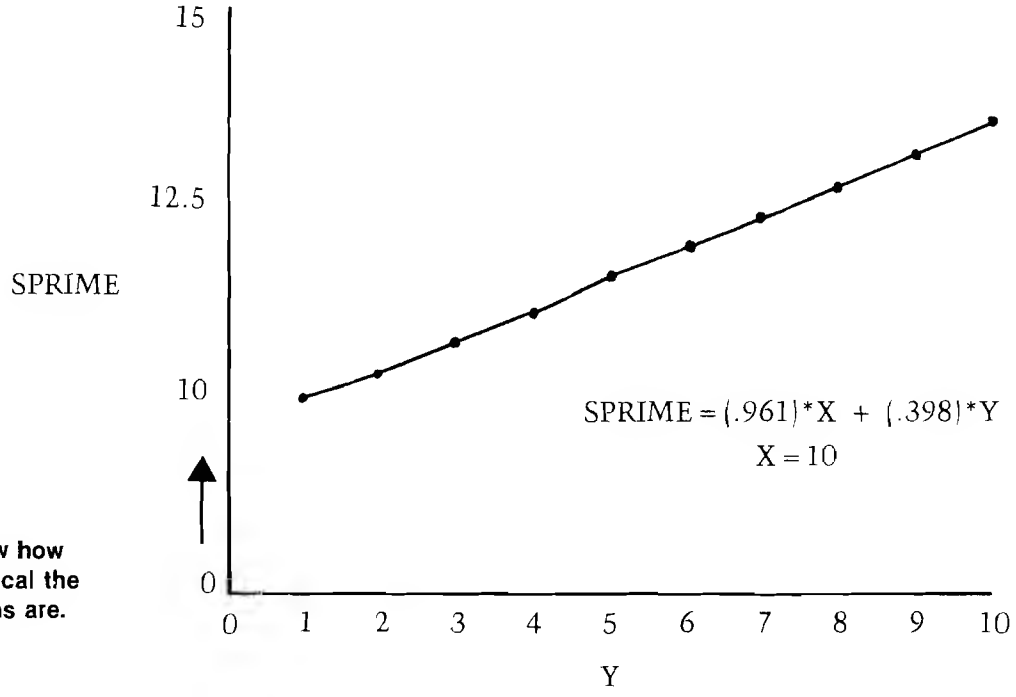


Figure 1. Graphs show how nearly identical the two functions are.

	S	SPRIME	ERROR (%)
1	10.0498756	10.008	.416678098
2	10.198039	10.406	2.03922505
3	10.4403065	10.804	3.48355178
4	10.7703296	11.202	4.00795888
5	11.1803399	11.6	3.7535541
6	11.6619038	11.998	2.88200115
7	12.2065556	12.396	1.55198879
8	12.8062485	12.794	.0956445798
9	13.4536241	13.192	1.94463627
10	14.1421356	13.59	3.90418847

Figure 2. Validity Program Outputs

```
 5  REM ERROR VALIDATION PROGRAM
 6  REM BY C WILLIAMS, 3/84
10  HOME : VTAB 1
20  PRINT "    S";: HTAB 15: PRINT      "SPRIME";:
    HTAB 28: PRINT "ERROR (%)"
30  POKE 34,2: VTAB 3
40  FOR Y = 1 TO 10.
50  X = 10.
60  S = ((X ↑ 2.) + (Y ↑ 2.))  0.5
70  SPRIME = (0.961 * X) + (0.398 * Y)
80  DF = ABS ((S - SPRIME) / S) * 100.
90  PRINT Y;: HTAB 4: PRINT S;: HTAB 15:
    PRINT SPRIME;: HTAB 28: PRINT DF
100  NEXT
110  END
```

MICRO

FLOPPY DISKS SALE *$1.19 ea.

Economy Model or Cadillac Quality

LORAN CERTIFIED PERSONAL COMPUTER DISK ***We have the lowest prices!*** LORAN CERTIFIED PERSONAL COMPUTER DISK

***ECONOMY DISKS**

Good quality 5¼" single sided single density with hub rings.

Bulk Pac	100 Qty.	$1.19 ea.	Total Price	$119.00
	10 Qty.	1.39 ea.	Total Price	13.90

CADILLAC QUALITY (double density)

- ***Each disk certified***
- ***Free replacement lifetime warranty***
- ***Automatic dust remover***

For those who want cadillac quality we have the Loran Floppy Disk. Used by professionals because they can rely on Loran Disks to store important data and programs without fear of loss! Each Loran disk is 100% certified (an exclusive process) plus each disk carries an exclusive FREE REPLACEMENT LIFETIME WARRANTY. With Loran disks you can have the peace of mind without the frustration of program loss after hours spent in program development.

100% CERTIFICATION TEST

Some floppy disk manufacturers only sample test on a batch basis the disks they sell, and then claim they are certified. Each Loran disk is individually checked so you will never experience data or program loss during your lifetime!

FREE REPLACEMENT LIFETIME WARRANTY

We are so sure of Loran Disks that we give you a free replacement warranty against failure to perform due to faulty materials or workmanship for as long as you own your Loran disk.

AUTOMATIC DUST REMOVER

Just like a record needle, disk drive heads must travel hundreds of miles over disk surfaces. Unlike other floppy disks the Loran smooth surface finish saves disk drive head wear during the life of the disk. (A rough surface will grind your disk drive head like sandpaper). The lint free automatic CLEANING LINER makes sure the disk-killers (dust & dirt) are being constantly cleaned while the disk is being operated. PLUS the Loran Disk has the highest probability rate of any other disk in the industry for storing and retaining data without loss for the life of the disk.

Loran is definitely the Cadillac disk in the world

Just to prove it even further, we are offering these super LOW INTRODUCTORY PRICES

List $4.99 ea. INTRODUCTORY SALE PRICE $2.99 ea. (Box of 10 only) Total price $29.90

$3.33 ea. (3 quantity) Total price $9.99

All LORAN disks come with hub rings and sleeves in an attractive package.

DISK DRIVE CLEANER $19.95

Everyone needs a disk drive doctor

FACTS

- 60% of all drive downtime is directly related to poorly maintained drives.
- Drives should be cleaned each week regardless of use.
- Drives are sensitive to smoke, dust and all micro particles.
- Systematic operator performed maintenance is the best way of ensuring error free use of your computer system.

The Cheetah disk drive cleaner can be used with single or double sided 5¼" disk drives. The Cheetah is an easy to use fast method of maintaining efficient floppy diskette drive operation.

The Cheetah cleaner comes with 2 disks and is packed in a protective plastic folder to prevent contamination.

List $29.95 / Sale $19.95 * **Coupon $16.95**

Add $3.00 for shipping, handling and insurance. Illinois residents please add 6% tax. Add $6.00 for CANADA, PUERTO RICO, HAWAII, ALASKA, APO-FPO orders. Canadian orders must be in U.S. dollars. WE DO NOT EXPORT TO OTHER COUNTRIES.

Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery, 2 to 7 days for phone orders, 1 day express mail!

VISA — MASTER CARD — C.O.D.

No C.O.D. to Canada, APO-FPO.

PROTECTO ENTERPRIZES WE LOVE OUR CUSTOMERS!

BOX 550, BARRINGTON, ILLINOIS 60010

Phone 312/382-5244 to order

BIG FOUR

NEW 128K —MEGA BYTE DUAL DISK DRIVE—80 COLUMN

COMPUTER SYSTEM SALE!

HOME • BUSINESS • WORD PROCESSING

List Price $3717.95

LOOK AT ALL YOU GET FOR ONLY **$895.**

	LIST PRICE
① B128 COMMODORE 128K 80 COLUMN COMPUTER	$ 995 00
② 8050 DUAL DISK DRIVE (over 1 million bytes)	1795 00
③ 4023 100 CPS 80 COLUMN BIDIRECTIONAL PRINTER	499 00
④ 12" HI RESOLUTION 80 COLUMN GREEN OR AMBER MONITOR	249 00
• BOX OF 10 LORAN LIFETIME GUARANTEED DISKS	49 95
• 1100 SHEETS FANFOLD PAPER	19 95
• ALL CABLES NEEDED FOR INTERFACING	102 05
TOTAL LIST PRICE	$ 3717.95

$895. SALE PRICE

Printer replacement options (replace the 4023 with the following at these sale prices)

	LIST	SALE
* Olympia Executive Letter Quality Serial Printer	$ 699 00	$ 399.00
* Comstar Hi Speed 160 CPS 15½" Serial Business Printer	$ 779 00	$ 499.00
* Telecommunications Deluxe Modem Package	$ 199 00	$ 139.00

Plus You Can Order These Business Programs At Sale Prices

	LIST	SALE		LIST	SALE
Professional 80 Column Word Processor	$149 95	$99 00	Payroll	$149 95	$99 00
			Inventory	149 95	99 00
Professional Data Base	149 95	99 00	General Ledger	149 95	99 00
Accounts Receivable	149 95	99 00	Financial Spread Sheet	149 95	99 00
Accounts Payable	149 95	99.00	Program Generator	149 95	99.00

15 DAY FREE TRIAL We give you 15 days to try out this SUPER SYSTEM PACKAGE!! If it doesn't meet your expectations just send it back to us prepaid and we will refund your purchase price!!

90 DAY IMMEDIATE REPLACEMENT WARRANTY If any of the SUPER SYSTEM PACKAGE equipment or programs fail due to faulty workmanship or material we will replace it IMMEDIATELY at no charge!!

Write or Call For Free Catalog and Spec Sheets!!

Add $50.00 for shipping and handling!!
$100.00 for Canada, Puerto Rico, Hawaii orders.
WE DO NOT EXPORT TO OTHER COUNTRIES
Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery, 2 to 7 days for phone orders, 1 day express mail! Canada orders must be in U.S. dollars. We accept Visa and MasterCard. We ship C.O.D. to U.S. addresses only.

PROTECTO ENTERPRIZES (WE LOVE OUR CUSTOMERS)
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

SANYO MONITOR SALE!!

9" Data Monitor

- 80 Columns x 24 lines
- Green text display
- Easy to read - no eye strain
- Up front brightness control
- High resolution graphics
- Quick start - no preheating
- Regulated power supply
- Attractive metal cabinet
- UL and FCC approved

12" Screen Amber or Green Text Display $99

- *15 Day Free Trial - 90 Day Immediate Replacement Warranty*

12" Hi-Resolution Amber or Green Screen Monitor $119.00
this is a 1000 Line, 80 Column, High Resolution Monitor with crisp clear text that is easy to read! A must for Word Processing! Includes special Software Discount coupon.
List $249.00 **SALE $119.00**

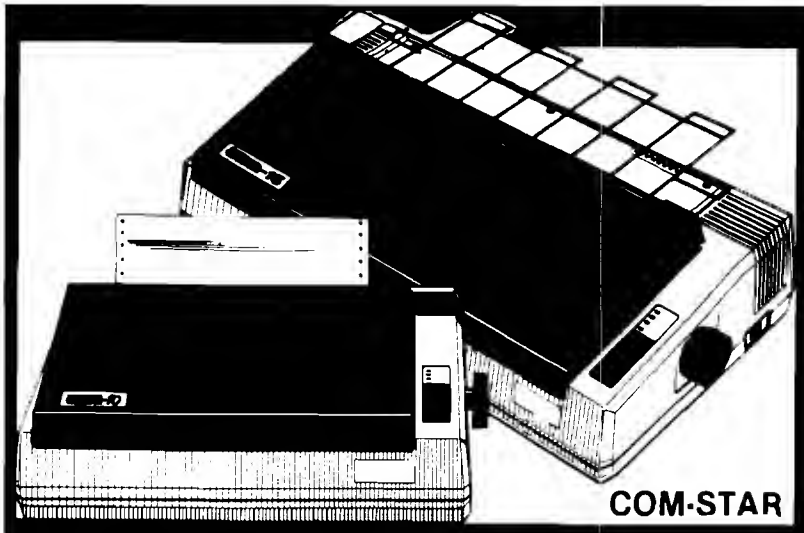
14" Hi-Resolution Color Monitor $229.00
This 14" color monitor has the sharpest and clearest resolution of any color monitor we have tested! Beautiful color contrast! Also compatible with video recorders. Includes special Software Discount coupon.
List $399.00 **SALE $229.00 (IBM Compatable)**

• LOWEST PRICES • 15 DAY FREE TRIAL • 90 DAY FREE REPLACEMENT WARRANTY
• BEST SERVICE IN U.S.A. • ONE DAY EXPRESS MAIL • OVER 500 PROGRAMS • FREE CATALOGS

Add $10.00 for shipping, handling and insurance. Illinois residents please add 6% tax. Add $20.00 for CANADA, PUERTO RICO, HAWAII, ALASKA, APO-FPO orders. Canadian orders must be in U.S. dollars. WE DO NOT EXPORT TO OTHER COUNTRIES.
Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery. 2 to 7 days for phone orders. 1 day express mail!
VISA — MASTER CARD — C.O.D

PROTECTO
ENTERPRIZES (WE LOVE OUR CUSTOMERS!)
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

FANTASTIC COMPUTER PRINTER SALE!!



COM-STAR T/F

Tractor
Friction
Printer

only $ 169**

- Lowest Priced, Best Quality, Tractor-Friction Printers in the U.S.A.
- Fast 80-120-160 Characters Per Second • 40, 46, 66, 80, 96, 132 Characters Per Line Spacing
- Word Processing • Print Labels, Letters, Graphs and Tables • List Your Programs
- Print Out Data from Modem Services • "The Most Important Accessory for Your Computer"

** DELUXE COMSTAR T/F 80 CPS Printer — $169.00

This COMSTAR T F (Tractor Friction) PRINTER is exceptionally versatile It prints 8½" x 11" standard size single sheet stationary or continuous feed computer paper Bi directional, impact dot matrix, 80 CPS, 224 characters (Centronics Parallel Interface)

Premium Quality 120-140 CPS 10" COM-STAR PLUS+ Printer $269.00

The COM-STAR PLUS+ gives you all the features of the COMSTAR T F PRINTER plus a 10" carriage, 120 140 CPS, 9 x 9 dot matrix with double strike capability for 18 x 18 dot matrix (near letter quality), high resolution bit image (120 x 144 dot matrix), underlining, back spacing, left and right margin settings, true lower decenders with super and subscripts prints standard, italic block graphics and special characters It gives you print quality and features found on printers costing twice as much!! (Centronics Parallel Interface) (Better than Epson FX80) List $499 00 **SALE $269.00**

Premium Quality 120-140 CPS 15½" COM-STAR PLUS+ Business Printer $379.00

Has all the features of the 10" COM-STAR PLUS+ PRINTER plus 15½" carriage and more powerful electronics components to handle large ledger business forms! (Better than Epson FX 100) List $599 **SALE $379.00.**

Superior Quality 140-160 CPS 10" COM-STAR PLUS+ IBM Pers/Bus Printer $389.00

Has all the features of the 10" COM-STAR PLUS+ PRINTER! It is especially designed for all IBM personal computers! 140 160 CPS HIGH SPEED PRINTING 100% duty cycle, 2K buffer, diverse character fonts, special symbols and true decenders, vertical and horizontal tabs.
A RED HOT IBM personal business printer at an unbelieveable low price of $389 00 plus one year immediate replacement warranty (centronics parallel interface) List $699 **SALE $389.00**

Superior Quality 160-180 CPS 10" COM-STAR PLUS+ Business Printer $399.00

This SUPER HIGH SPEED COM-STAR PLUS + PRINTER 160-180 CPS has a 10" carriage with all the COM STAR PLUS+ features built in! It is especially designed with more powerful electronics to handle larger ledger business forms! Exclusive bottom feed! (Centronics parallel Interface) also compatable with all IBM Personal/Business Computors!! One year immediate replacement warranty
List $699 **SALE $399**
15½" Printer List $799 **SALE $499.**

Olympia
Executive Letter Quality DAISY WHEEL PRINTER $399.00

This is the worlds finest daisy wheel printer **Fantastic Letter Quality.** up to 20 CPS bidirectional, will handle 14 4 forms width! Has a 256 character print buffer, special print enhancements built in tractor feed (Centronics Parallel and RS232C Interface) List $699 **SALE $399.**

PARALLEL INTERFACES

For VIC-20 and COM 64 — $49 00 For Apple computers — $79 00 Atari 850 Interface — $79 00 For ALL IBM Computers — $89 00

•15 Day Free Trial -180 Day Immediate Replacement Warranty

Add $14 50 for shipping, handling and insurance. WE DO NOT EXPORT TO OTHER COUNTRIES EXCEPT CANADA.

Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery, 2 to 7 days for phone orders, 1 day express mail! Canada orders must be in U.S. dollars. VISA — MASTER CARD ACCEPTED. We ship C.O.D.

PROTECTO
ENTERPRIZES (WE LOVE OUR CUSTOMERS)
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

COM-STAR PLUS+ Print Example: ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890

Commodore - 64

WORD PROCESSING BREAKTHROUGH!

SCRIPT-64 EXECUTIVE WORD PROCESSOR

(80 Columns in Color)

40 or 80 columns in color or black and white; turns your computer into a Business Machine!

Rated best by COMMODORE. This is the finest word processor available. Features include line and paragraph insertion/deletion, indentation, right and left justification, titles, page numbering, characters per inch, etc. All features are easy to use and understand. With tabs, etc. SCRIPT-64 even includes a 250 word dictionary/spelling checker to make sure your spelling is correct. The dictionary is user customizable to any technical words you may use. Furthermore, all paragraphs can be printed in writing and everyday letters are a snap. To top things off, there is a 100 page manual and help screens to make learning how to use SCRIPT-64 a snap. This word processor is so complete we can't think of anything it doesn't have. When combined with the complete database you have a powerful mailmerge and label program that lets you customize any mailing list with personalized letters. List $99.95. **Sale $59.00.** *Coupon Price $49.00. (Disk only.)

SCRIPT-64 20,000 WORD DICTIONARY

Allows you to check spelling on 20,000 most often mispelled words! List $29.95. **Sale $19.95.** *Coupon Price $12.50 (Disk only.)

SCRIPT-64 COMPLETE DATABASE

(Plus Mail Merge and Labels)

This powerful DATABASE is user friendly and makes any information easy to store and retrieve. The user defines the fields and then can add, change, delete, and search for any category wanted! Must be used with the SCRIPT-64 EXECUTIVE WORD PROCESSOR. When combined with the Executive Word Processor you can search out any category (zip codes, even hair color, etc.) and print super personalized letters! 600 names can be sorted and formulated on each disk in any order or category! Will handle any size mailing list by changing or adding disks! List $69.00. **Sale $39.00.** *Coupon Price $29.00.

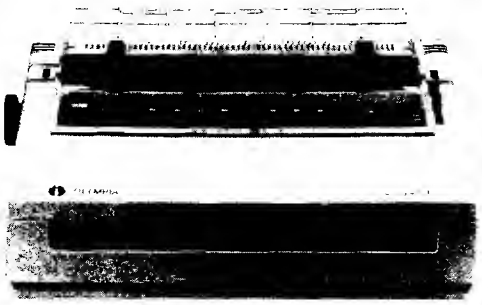
• LOWEST PRICES • 15 DAY FREE TRIAL • 90 DAY FREE REPLACEMENT WARRANTY
• BEST SERVICE IN U.S.A. • ONE DAY EXPRESS MAIL • OVER 500 PROGRAMS • FREE CATALOGS

WE SHIP C.O.D. HONOR VISA AND MASTER CHARGE
ADD $3.00 SHIPPING FOR C.O.D. ADD $2.00 MORE
SPECIAL SERVICES:
One Day — Express Mail add $10.00

PROTECTO ENTERPRIZES (WE LOVE OUR CUSTOMERS)
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

Olympia

EXECUTIVE LETTER QUALITY "DAISY WHEEL PRINTERS"

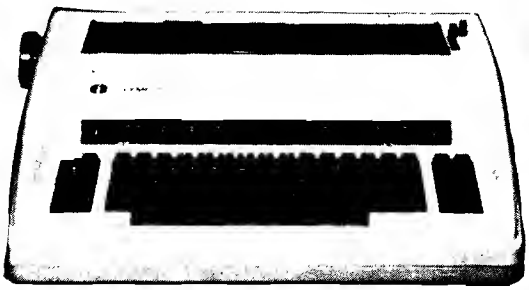


Executive Letter Quality Printer

World's Finest Computer Printer

List Price $699 **SALE $399**

- Daisywheel printer, bidirectional with special print enhancements.
- Print speed up to 20 characters per second.
- 10, 12, and 15 characters per inch.
- 256 character print buffer.
- 14.4" forms width.
- Print line width: 115, 138, and 172 characters.
- Serial RS-232-C and parallel Centronics interface ports built-in.
- Built-in bidirectional forms tractor.
- Operating status control panel.



Executive Letter Quality Printer/Typewriter

World's Finest "Combination" Printer/Typewriter

List Price $799 **SALE $489**

- Superb computer printer combined with world's finest electronic typewriter!
- Better than IBM selectric — used by world's largest corporations!
- Two machines in one — just a flick of the switch!
- Superb letter quality correspondence — home, office, word processing!
- Extra large carriage — allows 14-1/8" paper usage!
- Drop in cassette ribbon — express lift off correction or eraser up to 46 characters!
- Precision daisy wheel printing — many type styles!
- Pitch selector — 10, 12, 15 CPS, Automatic relocate key!
- Automatic margin control and setting! Key in buffer!
- Electronic reliability, built in diagnostic test!
- Centronics parallel interface built-in

15 Day Free Trial - 90 Day Immediate Replacement Warranty

COM 64 — VIC-20 INTERFACE	$59.00
APPLE INTERFACE	$79.00

Add $17.50 for shipping, handling and insurance. Illinois residents please add 6% tax. Add $35.00 for CANADA, PUERTO RICO, HAWAII, ALASKA, APA-FPO orders. Canadian orders must be in U.S. dollars.
WE DO NOT EXPORT TO OTHER COUNTRIES.

Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days delivery, 2 to 7 days for phone orders, 1 day express mail!

VISA — MASTERCARD — C.O.D.

No C.O.D. to Canada, APO-FPO

PROTECTO ENTERPRIZES (WE LOVE OUR CUSTOMERS)
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

by Ralph Tenny
Richardson, Texas

We're continuing with the design of an output adapter for the expansion port of a Radio Shack Color Computer. This adapter will interconnect an Epson MX-80 printer, a Commodore 64, a 32k Color Computer and a 64k Color Computer. The two 64k computers will input to the 32k Color Computer, which will serve as a printer buffer for both the other machines. The interface card will plug into the 32k machine and perform all the interface functions needed except for power sensing. A smart power box will sense when either computer is turned on, and power up the 32k machine. This machine must then self-boot and begin sensing when either computer sends data to be printed.

Last month's column gave a set of specifications for the four ports needed to accomplish the interfacing. These ports are:

1. Parallel input from the Commodore. This input actually comes from a The Connection serial-parallel converter currently used with the Epson/Commodore combination. This choice was made to insure continued compatibility with all Commodore software currently being used.
2. Parallel output to the printer.
3. Serial input from the 64k Color Computer.
4. Serial output (unassigned).

Figure 1 shows the schematic of the interface board. Ports 1 and 2 are implemented using a 6522 Versatile Interface Adapter, which gives two 8-bit I/O ports, two 16-bit timers, automatic input/output handshake, and synchronous serial communication. Each of the major functions can issue an interrupt, and a separate interrupt input is associated with each port.

The serial communications will be performed by a 6850 ACIA with switch-selectable baud rates of 300, 600, 1200 and 2400. This device is a programmable UART (Universal Asynchronous Receiver/Transmitter) which furnishes status output and input lines capable of managing I/O handshaking. Both receiver and transmitter sections can issue interrupts.

The specifications also call for a Busy signal capability on both parallel ports. This is required to be sure of compatibility with any printer or other parallel input or output device which may be used to drive it. The Busy signals will be programmed to be compatible with Epson and similar printers. The output Busy signal (used on the parallel input port) is held in a 74LS75 4-bit latch which connects to the upper nibble on the CoCo data bus. The Busy input from the parallel output port is gated onto the data bus with a 74LS126 4-bit tri-state buffer. Both the latch and the tri-state buffer have three unassigned channels which could be used for any single-bit I/O desired.

Baud rate generation was discussed in detail last time, except that the baud selection switches were omitted from the illustration. The corrected version and proper pin connections are shown in Figure 1 (U1 and U2). These two ICs form a programmable counter which resets itself each time the output pattern conforms to the bit pattern programmed into the switches.

The address decoding design was mentioned last time also. The circuit shown and the description given last time was incomplete. It also seems best to use three device SELECT signals instead of four. The reason is that the original SELECT signal for the parallel ports conflicts with disk port address space. This design is not intended for use with a disk, but could be if one of the several multi-pack interface units was in use. So, the design shown in Figure 1 has three SELECT signals: $FF50 for the parallel ports, $FF60 for the serial ports and $FF70 for the BUSY latch and BUSY flag input.

In addition to the SELECT signals, certain other decoding must be done. The 74LS75 quad latch is not a bus-oriented device, so it has no SELECT input. All it has is an active-high GATE pin which allows the output to follow the input as long as GATE is high. The latch must be forced to capture only that data *written* to $FF70. The SELECT signal is active-low, so it must be inverted to properly gate the latch. Also, the Read/Write* (R/W*) signal must be used to ensure that only WRITE data is captured, that is, when R/W* is low. One section of U5 pulses the GATE line high *only* when both SELECT and R/W* are low.

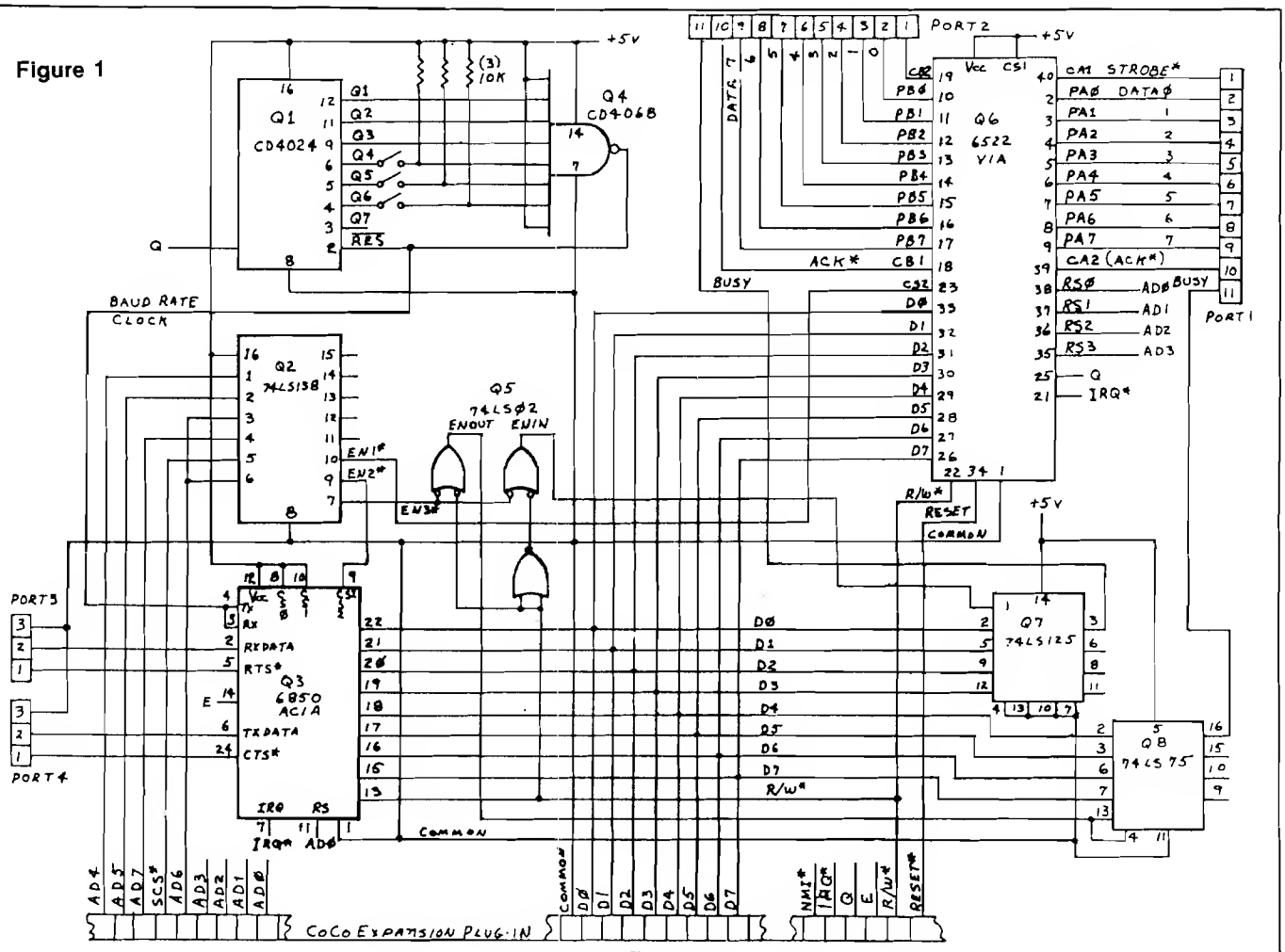
The 74LS125 tri-state gate also has no SELECT, and connects the input to the bus when the four (one for each section) enable lines are high. Also, this must happen only when the CPU is trying to *read* the data bus. So, a section of U5 inverts R/W* to enable a third section of U5. When both SELECT and (R/W*)* are low, the BUSY signal from the printer is allowed onto the data bus.

Additional decoding is necessary for both the ACIA and the VIA. Both have multiple registers, and REGISTER SELECT (RS) inputs The 6522 VIA has four RS inputs which are connected to address lines AD0-AD3, so that 16 internal registers can be selected. Almost all these registers can be read and written, just like normal memory locations. The ACIA has four registers that occupy only two memory addresses. Only one RS line is used (driven by AD0), which means that there are two Read Only registers and two Write Only registers. This causes extra programming overhead, which will be discussed when we have hardware ready to program!

Certain other circuit features and possibilities need to be discussed. First, the interrupts generated on the ACIA and VIA are shown connected to the IRQ* input. There could be a conflict with some CoCo software, so if this device is used as a general I/O board on an active computer, this should be moved to the NMI* input. Both the ACIA and VIA maintain an internal record of which section caused an interrupt, so each needs to be polled to determine which device caused the interrupt.

The VIA contains two counters which could have been programmed to make the proper clock frequencies for the ACIA. However, the normal hardware output for the counters are I/O lines already dedicated to parallel I/O. Another alternative exists. The counters will cause an interrupt, so the interrupt service routine could force a *read* of (for example) $FF30. The keyboard PIA in CoCo will respond, but so will pin 13 of U2. This decode strobe

Figure 1



could be used to toggle a flip-flop at twice the period of the desired baud rate clock.

This timer-generated baud clock would only be practical on a dedicated machine such as I will be using. Also, it is advisable to put the VIA on NMI* and the ACIA on IRQ*. Only the timer interrupt is time critical, but the STROBE* line should have reasonably prompt response to maintain high throughput for input data. The ACIA buffers a second character and will assert BUSY if necessary. However, the data throughput on even 2400 baud serial will not be greatly affected by a short BUSY hold on transmission.

Two other loose ends: If you should desire to have full eight-bit parallel input and output at $FF70, it is possible to use an octal latch in place of U8 and an octal tri-state buffer in place of U7. The required decoding is slightly different, and will be shown in next month's column.

The second loose end is that I promised to examine the bus loading in detail to decide if this interface board needed to be buffered from the innards of CoCo. The answer is yes; on two counts. First, the power loading (drive current furnished by the 6809 CPU) on the address lines is approaching the maximum specified value. This loading will reduce the bus's ability to drive a capacitive load at normal speed. In the second place, the capacitive loading for the address bus is very close to, or exceeding, the rated maximum. Therefore, this board *does* need buffering. The calculation, circuitry and other considerations will be presented next time also. MICRO

PEEK A BYTE™64

AN ESSENTIAL DISK & MEMORY UTILITY FOR THE COMMODORE 64™ & DRIVE

EASY TO USE - HELP - KEYSTROKE COMMANDS

- Disk Track/Sector Editor
  - Examine and modify disk sector data
  - File Follower - memory for 151 sectors
  - Fast 1541 disk compare and error check
- Display Memory and Disk Data in Hex, ASCII or Screen Code
- Edit full page in Hex or ASCII
- Disassemble memory and disk data
- Search for string
- Un-new Basic pgms
- Read drive memory
- Convert Hex/Dec
- Free sector map
- Use DOS wedge
- Run ML routines
- Extensive manual
- Printer screen dump (serial bus)
- Fast machine code! Compatible with many Basic and monitor programs

DISKETTE CAN BE BACKED UP!!

QUANTUM SOFTWARE
P.O. BOX 12716, Dept. 64
LAKE PARK, FL 33403

ALL FOR $29.95 US Post Paid

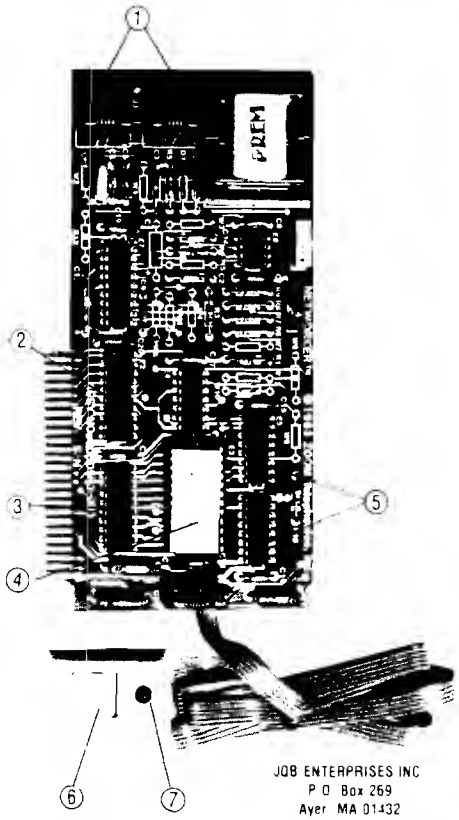
TO ORDER: Send check or money order, US dollars
Florida residents add 5% sales tax
COD add $2. Call 305-840-0249

Commodore 64 is a registered tradmark of Commodore Electronics Ltd.
PEEK A BYTE is a trademark of Quantum Software

FREE! NETWORKER NOW INCLUDES A FREE SUBSCRIPTION TO The Source®

Complete Apple Modem $129

Single-Slot 300 Baud Direct-Connect Modem for Apple II, II+, IIe and Franklin computers



FEATURES OF THE NETWORKER™

1. DIRECT CONNECT – No acoustic coupling needed. Two modular telephone jacks – one for phone – one for line
2. SINGLE CHIP MODEM for greater reliability
3. ON BOARD FIRMWARE contains a terminal program
4. ON BOARD SERIAL INTERFACE – no extra cards to buy. Software selectable data format, 7 or 8 data bits, one or two stop bits, odd or even parity, full or half duplex
5. 300 BAUD software selectable for 110 baud
6. SWITCH CONTROL for answer originate sits next to keyboard
7. CARRIER DETECT LED gives you line status at a glance

ALL THIS PLUS

- COMPLETE with NETWORKER SOFTWARE to give you
  - Text trapping of entire display into RAM memory
  - Disk storage capability for all trapped text
  - On screen menu and status indicators
- FREE SUBSCRIPTION TO THE SOURCE, the popular dial up information system
- SOFTWARE COMPATIBILITY – with all common Apple communication software
- COMPATIBLE with both rotary and tone phones
- FCC APPROVED – Made in USA
- ONE YEAR MANUFACTURER'S WARRANTY

NETWORKER™ INCLUDES A COMPLETE PACKAGE

- Networker modem board and complete system
- Modular phone line cord
- Networker software on a disk ready to run
- Complete instruction manual

NETMASTER™ COMMUNICATIONS SOFTWARE

For $179 we include with the **NETWORKER** the **NETMASTER** Communications Software for advanced users. **NETMASTER** will let you transfer games, computer graphics programs, sales records, documents – in fact any Apple file of any size – to another computer, directly from disk to disk without errors, even through noisy phone lines.

For transfering information between computers **NETMASTER**'s superb error checking and high speed are an unbeatable combination. With a **NETMASTER** on each end, you can transfer information three to five times faster than other communications packages like Visiterm™ or ASCII™ Express. Error free.

Your best buy in modem history. The **Networker,**™ a plug-in single-slot direct connect modem for the Apple II family of computers. Send electronic mail to a friend or business associate, use your school's computer, access hundreds of computer bulletin boards or thousands of data bases for up-to-the-minute news, sports, weather, airline and stock information.

JQB ENTERPRISES INC
P.O. Box 269
Ayer, MA 01432

There's absolutely nothing else to buy. You get the modem board, communication software, and a valuable subscription to America's premier information service, **THE SOURCE**. For $129, it's an unbeatable value.

This is the modem that does it all, and does it for less. The Apple Communications Card is on board, so no other interface is needed. It's 300 baud, the most commonly used modem speed. And it comes complete with **NETWORKER** Communications Software on an Apple-compatible disk, giving you features no modem offers.

Like the ability to lock on-screen messages into your Apple's **RAM**, and then move the information onto a disk for easy reference and review. A terminal program that turns your computer into a communications command center with on-screen help menus, continuous updates of memory usage, carrier presence, and communication status.

But **NETMASTER**'s not stuffy. It will talk to those other communications packages, but they don't work as fast and they don't check errors like **NETMASTER**. And **NETMASTER** doesn't only work with the **NETWORKER** modem. Even if you already have another modem for your Apple, **NETMASTER** is an outstanding value in communications software, so we sell **NETMASTER** by itself for $79. **NETMASTER** requires 48k of RAM, one disk drive, and the **NETWORKER** or another modem.

WE EVEN GIVE YOU SOMEONE TO TALK TO!

Your purchase of the **NETWORKER** with or without **NETMASTER** comes complete with a membership to **THE SOURCE**, with its normal registration fee fully waived. **THE SOURCE** will put a world of electronic information and communication services at your fingertips – instantly. Electronic mail and computer conferencing. Current news and sports. Valuable business and financial information. Travel services. A wealth of information about personal computing. Even games. All fully compatible with your equipment, and ready to use at once.

To Order Call Toll Free

800-824-7888 Continental US
800-824-7919 Alaska and Hawaii
or anywhere in the world
916-929-9091
Ask for operator #592

MAIL ORDERS

PLEASE WRITE NUMBER OF ITEMS IN BOX

☐ NETWORKER $129 ☐ NETMASTER $79
☐ NETWORKER/NETMASTER COMBO $179

NAME _____
ADDRESS _____
CITY _____ STATE _____ ZIP _____

Mass. residents add 5% sales tax
Total Enclosed _____

VISA master charge C.O.D.
C.O.D. ORDERS ADD $3.00

☐ MASTERCARD ☐ VISA ☐ CHECK ☐ C.O.D.
CARD NUMBER _____ EXPIRES _____

SIGNATURE _____
(Credit Card orders must be signed)

Send Orders and Make Checks Payable to

JQB Enterprises Incorporated

P.O. BOX 269 AYER MASSACHUSETTS 01432
All Prices Quoted are for Prepaid Orders — Prices Subject to Change Without Notice

feature

68000 Exception Processing

by Mike Rosing
Littleton, Colorado

The 68000 microprocessor has many attributes found on mini and main frame computers. These include system and user modes, levels of interrupts and error recovery from bad software. Compared to 8 bit microprocessors like the 6502 and 6809, the 68000 seems incredibly complex.

Fortunately, the designers of the 68000 came up with a logical and straight forward method of handling all the complexities. The purpose of this article is to describe "exception processing" on the 68000. The following parts will give examples of how to take advantage of the 68000's capabilities. Part 2 will cover software exception processing in general; Part 3 will cover hardware exception processing for the Sage II.

68000 uses exception processing to handle software and hardware.

Status Register

Like other microprocessors, the 68000 has a status register. This holds the carry, overflow, zero and negative bits which are found on all microprocessors. The 68000 has additional bits called trace, interrupt mask and supervisor state. The bit positions within the status register are shown in Table 1.

Table 1 - Status Register

	Bit	Description
MSB	15	Trace Mode
	14	Unused
	13	Supervisor State
	12-11	Unused
	10-8	Interrupt Mask
	7-5	Unused
	4	Extend (X bit)
	3	Negative
	2	Zero
	1	Overflow
LSB	0	Carry

Bits 0-7 are called the user byte and bits 8-15 are called the system byte. The trace mode bit is useful for tracing programs one instruction at a time. The interrupt mask determines what level of interrupt can be processed, lower levels being ignored. We will get into those bits later. For now the supervisor state is most important.

If you have ever written a program on an 8 bit machine which accessed every byte in I/O space (like C000 to CFFF on the Apple 2) you will appreciate the separation of user state and supervisor state on the 68000. When the 68000 is in user state, it can not access the system byte of the status register. Nor can it access addresses which are specified to be in supervisor space.

Register A7 is also affected by the supervisor state bit. If the bit is set A7 points to the supervisor stack pointer. When the supervisor state bit is clear A7 points to the user stack pointer. Most systems keep these stacks in different areas of memory.

Exceptions

Once the 68000 goes to user state, it can not change the supervisor state bit. Unless an exception occurs, the 68000 will stay in user state. Since exceptions can be forced by software, this is not a problem. In fact, it ensures program integrity since exceptions are all outside the users normal needs.

Exceptions include interrupts, hardware errors, software errors and traps. Interrupts are caused by external devices. Hardware errors are part of external logic to the 68000. Software errors include division by zero and registers out of bounds. Traps are similar to a software generated interrupt.

All 68000 exceptions go thru four steps. Step one is to make a copy of the status register. This ensures that after the exception is handled the processor can return to its original state. The supervisor state bit is set putting the processor in supervisor mode and sets the stack pointer to the supervisor stack.

Step two determines the vector number of the exception. This vector is a pointer to the code which the 68000 will execute to take care of the exception. In some cases this number is placed on the bus by an external device. In other cases the vector number is generated by the 68000.

In step three, the program counter is pushed on the supervisor stack followed by the status register copy made in step one. If the exception is a bus or address error, more information will be pushed on the stack during this step.

Step four sets the program counter to the address found in step two and normal execution resumes.

The address pointers used by the 68000 in step two are located at addresses 0 thru $3FF. Every four bytes represents a 32 bit pointer. This is enough room for 256 pointers. Multiplying the vector number by four gives the address of the pointer. This in turn has the address of the code to execute.

The first 64 exception vectors have specific meanings. For example, vector number 5 is the divide by zero exception. Vectors 48 thru 63 are reserved for future use. Vectors 64 thru 255 are user definable.

Table 2 — Software Exception Vectors

Vector Number	Hex Address	Description
3	C	Address error
4	1Ø	Illegal instruction
5	14	Zero divide
6	18	CHK instruction
7	1C	TRAPV instruction
8	2Ø	Privilege violation
9	24	Trace
1Ø	28	Line 1Ø1Ø emulator
11	2C	Line 1111 emulator
32-47	8Ø-BC	TRAP instructions

Table 3 — Hardware Exception Vectors

Vector Number	Hex Address	Description
Ø	Ø	Reset
2	8	Bus error
15	3C	Uninitialized interrupt
24	6Ø	Spurious interrupt
25-31	64-7C	Autovector interrupts
64-255	1ØØ-3FC	User interrupts

Software Exceptions

Table 2 shows the vector numbers and locations for the software exceptions. A short description of each is given here. Actual examples will be given in Part 2 of this article.

Address error: Attempt was made to reference word or long word address on an odd boundary.

Illegal instruction: Attempt was made to execute data. Patterns $4AFA, $4AFB and $4AFC are "permanently illegal" according to Motorola.

Zero divide: Attempt to divide by zero using DIVU or DIVS instructions.

CHK instruction: The check instruction is used to compare a register against bounds. If not in proper range the exception occurs.

TRAPV instruction: Exception taken if overflow bit is set.

Privilege violation: Attempt was made in user mode to change system byte of status register.

Trace: If the trace bit is set in the status register, the exception is taken at the end of each instruction. This pointer should be set to a debugger or monitor.

Line 1010 and Line 1111 emulator: 68000 instructions which have $A or $F as the first nibble will come here. Motorola has a set of instructions defined for the 68020 such as floating point operations. The purpose of these traps is to enable emulation of the 68020 by the 68000 for downward compatability.

TRAP instructions: These can be compared to software interrupts of 8 bit microprocessors. There are 16 available traps on the 68000.

Hardware Exceptions

Table 3 shows the hardware vectors. I call them hardware because the support chips placed around the 68000 determine how these vectors are generated.

Bus error: When pin 22 goes low on the 68000 this exception is processed. Usually a system is designed so any attempt to access memory which does not exist will pull this low.

Uninitialized interrupt: If an MMU (memory management unit) is attached to the 68000 it can generate this vector.

Spurious interrupt: If bus error goes low while an interrupt is being processed this exception is taken.

Autovector interrupts: Designed for use with 6800 peripherals. Almost all systems built to date use these as the only interrupts.

User interrupts: Space for interrupts generated using "normal" interrupt processing. Very few manufacturers use this because autovectoring is much simpler.

To understand how to use the 68000 exception processing a few examples are necessary. Part 2 of this article will discuss software exceptions which can be programmed on any 68000 system. Examples of hardware exceptions will be given in Part 3. These examples will be for a specific computer but the main ideas are transportable to other machines.

Part 2: Software Exceptions

In the first part of this article, the 68000 exception processing was described in the theoretical sense. This

Listing 1

```
* initialize trap vector

    lea       trapzero,a0      get trap address
    move.l    a0,$8Ø           put into vector location

* execute software exception

    trap      #Ø

* go back to user mode

    move      #Ø,sr        clear all of status register
    rts

* code executed by trap #Ø instruction

trapzero:bset  #5,(sp)      set supervisor bit in old status word
    rte                         return from exception
```

section's purpose is to create a better understanding of the 68000 exception processing activity by using concrete examples.

When an exception occurs, the 68000 copies the status register, gets the vector number, pushes the program counter and previous status register on the supervisor stack and jumps to the address stored for this vector.

As an extremely simple example the code in Listing 1 is actually very powerful.

The first two lines in Listing 1 initialize vector number 32. The sixteen trap instructions use vectors 32+n where n is in the range zero to fifteen. These two lines of code point out the flexability of the 68000 exception system. The exception processing code may be anywhere in memory. It also points out how simple it is to change exception vectors.

The trap instruction is the same as a software interrupt instruction on a 6502 or 6809. The program counter and status register are pushed onto the supervisor stack. The number in the instruction determines which vector is chosen. Trap instructions are always immediate mode. The trap number is added to 32, then multiplied by 4. This gives the address of the trap code. In this case address $80 contains the address of trapzero.

The address stored in location $80 is put into the program counter. Execution now begins as usual. In this case, bit five of the byte pointed to by the supervisor stack is set. Remember that this is a copy of the status register. When the RTE is executed, the status register is pulled from the stack. Bit five happens to be the supervisor state bit. When the program counter is pulled from the stack we return to the instruction following the trap #0.

We have now gone from user mode to supervisor mode and the system is at our command. Depending on the operating system, this can be extremely dangerous. After playing with the system we may want to return to user mode. This is done by clearing the supervisor state bit in the status register (sr).

If an attempt were made at executing the move #0,sr in user mode the privilege violation exception (vector number 8) would occur. In this case the program counter pushed on the stack points to the instruction in violation, not the instruction afterwards. This is slightly different from the trap instruction.

Unimplemented instructions are similar to the privilege violation. The program counter pushed on the system stack points to the offending code. This is very useful as shown in the next example.

Suppose we have an application where 64 bits are required. As an example we create an instruction which has the following format:

Nibble	Hex value	Meaning
3	F	Line 1111 emulator
2	Ø	Addition
1	Ø,2,4,6	Register
Ø	Ø,2,4,6	pair

Nibble 3 forces exception processing on vector number 11 (address $2C). Nibble 2 can be used to specify one of 16 instructions. For this example only one is used. Nibble 1 specifies the source register pair. Since we want to add 64 bits, we will need two registers to hold the result. Zero means registers d0 and d1. Six means register d6 and d7. We will take odd numbers to be errors. Nibble 0 specifies the destination register pair.

Listing 2 shows how an emulator might be written. It first saves all the data registers. Errors are just ignored. Since any registers might be used, all are saved into memory. The source and destination register pairs are converted to memory offsets. These memory contents are then added. Notice that the *add with extend* instruction is not as flexable as the *add* instruction. This is a minor drawback of the 68000.

Initialization of the emulator requires the first line of Listing 2. This puts the address of the emulator into its appropriate vector.

Listing 2

```
* initialize line 1111 emulator and call it for testing

     move.l    line1111,$2c
     move      1,d2
     move      2,d4
     .dc.w     $fØ24
     rts

* emulator code for 64 bit addition of registers

line1111:  movem.l   dØ-d7/aØ,savezone   save data
           move.l    2(sp),aØ           point to instruction
           move      (aØ),dØ            get instruction
           btst      Ø,dØ          even destination?
           bne       error              nope
           btst      4,dØ          even source?
           bne       error              no again
           and       7,dØ          create
           lsl       2,dØ          destination offset
           move      (aØ),d1            create
           and       $7Ø,d1        source
           lsr       2,d1          offset
           lea       savezone,aØ        point to data
           move.l    4(aØ,d1),d2        get lower source bits
           add.l     d2,4(aØ,dØ)        add to lower destination bits
           move.l    (aØ,d1),d2         get high source bits
           move.l    (aØ,dØ),d3         get high destination bits
           addx.l    d2,d3              add with extend
           move.l    d3,(aØ,dØ)         reset destination
error:     movem.l   savezone,dØ-d7/aØ  restore new data
           add.l     2,2(sp)       point to next instruction
           rte                          return from exception
savezone:  .ds.l     9                  space for 9 longwords
           end
```

Execution is done by placing a word in the middle of normal code. Most assemblers require a construct shown in Listing 2 as .dc.w $F024.

When normal execution gets to this *instruction* exception processing takes over. Since the high nibble is $F = 1111 (base 2), vector number eleven is executed. This is just our emulator. Notice that we added 2 bytes to the program counter before returning from the exception. This prevents an infinite loop.

The trace exception is also a software interrupt. Like the trap instruction the program counter pushed on the stack points to the next instruction to be executed. This happens because the trace exception processing always occurs between instructions. To make effective use of the trace exception, one must know the I/O port addresses for efficient debugging.

Unlike the previous exceptions, the address error can occur anytime an effective address for a word or longword is on an odd byte. This is usually in the middle of an instruction execution. Because of this, more information is pushed on the supervisor stack during the third step of exception processing. This includes the instruction register which holds the first word of the offending instruction. The program counter is usually two to four bytes past this point. After the instruction register comes the effective address which is going to be odd. This is a longword. The lst word pushed on the stack includes the function code, a read/write bit and an instruction bit.

The function code corresponds to the 3 bits FCO, FC1 and FC2 on the 68000. These determine user or supervisor mode and program or data space. The read/write bit tells whether the access was during a read or write. The instruction bit tells if the error occured on an instruction. An address error on an instruction will occur on a line of code such as JMP (AO) where AO is odd. The jump instruction itself will be executed, but an address error will occur as soon as AO is transferred to the program counter.

To fully utilize these error exceptions, one has to send information to the programmer. This usually involves input and output. The final section of this article gives examples of hardware interrupts for I/O on a specific machine.

Part 3: Hardware Exceptions

The previous two parts of this article have described 68000 exception processing which was valid for any system. This final portion is specific to the Sage 2 because hardware exceptions are caused by the physical wiring connected to the 68000.

Hardware exceptions include reset, interrupts and bus error. Each of these are physically wired to the 68000. On the Sage, the reset is performed by power on, or by pressing the reset button on the back of the machine.

The bus error line on the Sage will be activated if the address strobe line is not released within two microseconds. Even the slowest EPROM's are faster than this. Usually one gets this error when attempting to access an address such as $4D696B65.

Interrupts are generated whenever any of the interrupt lines go low. All lines low indicates a level 7 interrupt. All lines high indicates no interrupt. The Sage is built with an LS148 priority encoder attached to the interrupt lines. This ensures that the 68000 sees only the highest level interrupt yet to be processed.

Reset Processing

During reset the 68000 looks for a system stack pointer at vector number 0. This is located at address 000000. It then reads the program counter from vector number 1 at address 000004. On the Sage these addresses are located in RAM. How does the Sage turn on at a known address?

When the reset line goes low a latch is cleared which relocates the monitor EPROMs to address 000000. The first longword in the EPROM is $400 which is the startup system stack. The next longword is $FE003C which becomes the program counter.

On the Sage, addressing the EPROMs at addresses in the $FE0000 range relocates the EPROMs to $FE0000. Thus, as soon as the program counter address hits the bus, the EPROMs are where they need to be and the 68000 is initialized.

The only way to change the reset system on the Sage is to burn new EPROMs. For most users this should be unnecessary.

Bus Error Processing

The bus error exception is exactly the same as the address error described in part two previously. It is proccssed whenever the bus error line (pin 22) goes low. To appreciate why this is useful in two microseconds on the Sage, remember that the 68000 is an asychronous device. DTACK has to go low before the address lines are released. The processor enters wait states until DTACK is returncd. It is perfectly happy to wait forever.

To avoid external circuitry which computes whether an address is valid it is much simpler to put on a timer. The timer is run at the start of each bus access and off when no bus access takes place. As long as the timer never runs out no bus error can occur. If no wait states are ever used, a complete bus access will require 250 nanoseconds and a complete instruction will require 500 nanoseconds (or more). A single wait state is 125 nanoscconds. Thus 16 wait states will go by before a bus error occurs.

This discussion is specific to the Sage only. Other systems will have different methods of generating bus errors. The access address is pushed on the stack in any case, and this is important to display for debugging purposes.

Interrupt Processing

According to the 68000 manual, normal interrupt processing requires the interrupting device to put its vector number on the lower data bus. This number should be greater than 64. This method allows for a total of 192 interrupts of any desired level. The implementation is not so easy in hardware. I don't know of any 68000 based machines which use this capability.

The lack of "normal" interrupts is due to the 68000 having autovector interrupts. By pulling the VPA line low (pin 21) during an interrupt acknowledge the 68000 looks at the interrupt level input lines (pins 23-25). This value is added to 24 to get the exception vector.

Level 7 is used for RAM parity errors. Levels 4, 5 and 6 are used for the IEEE port, the terminal input port and the floppy disk controller respectively. The level one interrupts are processed thru an Intel priority interrupt controller for eight interrupts. All

interrupts are autovectored and no provision was made for "normal" interrupt processing as defined by Motorola.

The eight level one interrupts include two clock timers, the terminal receive ready, the terminal and remote transmit ready, a ring detect, printer acknowledge and software generatable inputs. In order to give an example of interrupt processing on the 68000, we will mask all but the terminal USART generated interrupts.

The terminal interface chip is an Intel 8251A. This chip generates an interrupt whenever a character is received, as well as when it is ready to transmit a character.

The code shown in Listing 3 is very simple and should not be used for any real application because it does not save registers. It is a good example to follow though. The first line of code puts the processor into supervisor mode (see page 48, Listing 1). The mask on the next line prevents all interrupts except the USART transmit ready from entering the 8259. Next the autovector addresses are set to the interrupt codes. Registers are used directly for this example and are initialized. The 5 turns the USART on for transmission and reception of data. Finally, the stop instruction moves $2000 into the status register and halts processing.

When an interrupt occurs the processor executes code starting at the interrupt in question. If the USART is ready to transmit the termout interrupt will execute. If a character comes in from the keyboard, the termin code will execute. When the processor returns from the interrupts the code following the stop is executed.

To follow this, suppose the processor is halted. We press a key on the terminal which is transmitted to the USART. The USART causes interrupt level 5 and the 68000 autovectors to termin. The character is pulled from the USART, clearing the interrupt. We store it into the input buffer and bump the input buffer pointer. After returning from the interrupt, the 68000 checks to see if a character came in.

A character was received so D0 is not equal to D1. The character is moved to the output buffer and pointers are incremented. The transmitter is enabled and the processor halts again.

Since the transmitter is empty, it generates a level one interrupt via the 8259. The level one interrupt autovectors to termout. Since we set it up for only one possibility, the 8259 interrupt level is ignored. However, in a real system we would then have to pick the correct place to go to handle the interrupt.

The character is moved from the output buffer to the transmitter of the USART. The get pointer is bumped and the interrupt is ended.

At this point D0 equals D1 so we return to the stop instruction. The USART generates a transmitter empty interrupt so we again vector to the termout routine. At this point D2 equals D3 meaning the buffer is empty (or overflowed!). The endout point turns the transmitter enable off. Return from interrupt again finds D0 and D1 equal putting us at the stop instruction.

Even at 19.2K baud the transmission of characters is 100 times slower than the processor. This is only an example to show how interrupts can be used to run I/O in background while 68000 processes data in foreground.

To summarize, remember that exceptions all have vectors which point to the code which handles them. Any exception vector can be changed on any computer to suit the needs of your program.

Listing 3

```
*  Assume baud rates set up and 8259 initialized
         trap     #Ø                    go to supervisor mode
         move.b   #%11Ø11111,$FFCØ43    mask off all interrupts
*                                            except terminal transmit
 transmit
         move.l   #termout,$64          autovector 1
         move.l   #termin,$74           autovector 5
         lea      inbufr,aØ                pointer to input buffer
         lea      outbufr,a1               pointer to output buffer
fer
         moveq    #Ø,dØ                 set
         moveq    #Ø,d1                 up
         moveq    #Ø,d2                 get and
         moveq    #Ø,d3                 put pointers
on:      move.b   #5,#FFCØ73         enable transmission and
                                            reception
wait:    stop     #$2ØØØ                wait for interrupt
         cmp.b    dØ,d1                    char came in?
         beq      wait                     no,went out
         move.b   (aØ,dØ),(a1,d3)          move from input to output
put
         addi.b   #1,dØ                 bump input get pointer
         addi.b   #1,d3                 bump output put pointer
         bra      on                       enable transmission

*  terminal output interrupt handler
termout: move.b   #$C,$FFCØ41           set up 8259 for read
         move.b   $FFCØ41,d4                get interrupt level
         move.b   #$2Ø,$FFCØ41          clear interrupt
         cmp.b    d2,d3                    hit end of
         beq      endout                   data to transmit?
         move.b   (a1,d2),$FFCØ71          send out char
         addi.b   #1,d2                 bump get pointer
         rte                                end of interrupt
endout:  move.b   #4,$FFCØ73            disable transmission
         rte

*  terminal input interrupt handler
termin:  move.b   $FFCØ721,d4               get input char
         move.b   d4,(aØ,d1)                put into input buffer
         addi.b   #1,d1                 bump put pointer
         cmp.b    dØ,d1                     hit get pointer?
         bne      endin                     yes, problems
         move.b   #Ø,$FFCØ73            so disable receive
endin:   rte                                end of interrupt
```

Transferring dBase II Files For Use With Wordstar/Mailmerge

by Robert R. Carroll
Woodland Hills, California

Alter your dBase II files and use them to produce personalized letter forms.

When using the popular programs dBASE II and Wordstar/Mailmerge in business, an opportunity often presents itself where the two can be used together to produce personalized form letters. On the surface, it would seem that this should be no problem; however, there exists certain peculiarities in each program that forces a bit of thought as to how to manipulate dBASE II files so that Mailmerge sees just what it needs to perform correctly with all types of data. If you're not careful, you will undoubtedly find form letters loaded with unwanted spaces due to the trailing blanks leftover from the fixed field lengths of your data from dBASE II's .DBF (data) files. Or worse, embedded commas in your data will confuse Mailmerge to the point where it won't track your data correctly. In either case your letters will look far from personal. This article will explore a simple yet effective method for flawlessly transferring dBASE II data files for this use.

Of the many books and articles written on these programs, none has come up with a simple, easy-to-use method of transferring the data files. Typical schemes require writing an external BASIC program or using a word processor in a lengthy, hard-to-remember procedure. dBASE II provides all the functions necessary to make the transfer complete without resorting to difficult external measures. Besides, many users of these programs don't have the time, inclination or programming knowledge to use them.

Now, let's look at a sample dBASE II .DBF file called DATA.DBF which contains mailing list data. Using dBASE II's LIST STRUCTURE command we see the following:

```
. USE DATA
. LIST STRUCTURE
STRUCTURE FOR FILE:  A:DATA.DBF
NUMBER OF RECORDS:   ØØØØ3
DATE OF LAST UPDATE: Ø3/15/84
PRIMARY USE DATABASE
```

FLD	NAME	TYPE	WIDTH	DEC
ØØ1	MRMRS	C	ØØ4	
ØØ2	FIRSTNAME	C	Ø15	
ØØ3	LASTNAME	C	Ø15	
ØØ4	COMPANY	C	Ø2Ø	
ØØ5	STREET	C	Ø2Ø	
ØØ6	CITY	C	Ø15	
ØØ7	STATE	C	ØØ4	
ØØ8	ZIP	C	ØØ6	
** TOTAL **			ØØØ98	

We can see that each field has a specific width determined by the user at the time the database was created. dBASE II will reserve this width for the data when it is stored on disc, whether the actual data fills up the entire width or not. For example, a LASTNAME of Deltawashington fills up our field width of 15 characters quite nicely. However, Stein leaves us with 10 trailing blanks which will be carried right into our form letter if we don't get rid of them somehow. Let's enter some sample data to clarify the example.

```
Rec #   MRMRS FIRSTNAME   LASTNAME        COMPANY
00001   Mr.   John H.      Stein           Tate and Sons
00002   Mrs.  Joanne T.    Houseman        Melt, Inc.
00003   Ms.   Toni J.      Deltawashington Lewis, Jones, and Co.

STREET              CITY         STATE   ZIP
24 Lake Avenue      Chicago      IL      60606
970 Tulips Street   New York     NY      10019
11678 Riverside Dr. Burbank      CA      91356
```

Now suppose we want to use this data to create a personalized form letter called FORMLTR written using Wordstar/Mailmerge. The letter might read as follows:

```
..FORMLTR
.op (to omit page number in printout)
.df DATATEXT.TXT (datafile created from dBASE II file DATA.DBF)
.rv MRMRS,FIRSTNAME,LASTNAME,COMPANY,STREET,CITY,STATE,ZIP
(tells Mailmerge to read values in specific order)

Dear &MRMRS& &LASTNAME&:

We regret to inform you that we cannot use &COMPANY& in this
year's exhibition. Thank you for your interest.

Sincerely,

Clyde T. Newshandler

.pa(to begin new page)
```

If we don't remove the trailing blanks we would get something like:

```
Dear Mr.  Stein          :

We regret to inform you that we cannot use Tate and Sons      in
this year's exhibition. Thank you for your interest.
```

The blanks in the text of the letter automatically tell the reader a computer is talking at him. Also, Mailmerge must see its data separated by commas and if a piece of data has a comma embedded in it, the entire piece must be enclosed in quotes. For example, in our Rec # 00002, the COMPANY field contains Melt, Inc. Mailmerge must see "Melt, Inc." so that it doesn't confuse the embedded comma with a field separator and think that "Inc." is the next data field. In this unfortunate occurrence, the tracking of all the data would be one field off for the rest of the letters generated. Thus we might get:

```
Dear 10019 Ms.:

We regret to inform you that we cannot use Toni J. in this .......
```


Such A Deal Inc.

LEGEND
PERIPHERAL PRODUCTS

DOT MATRIX PRINTERS
80 CPS $239
100 CPS $259
120 CPS $299
150 CPS $349

MicroPro®
WordStar $269
Pro Pack $369

Novation

APPLE CAT II
300 Baud, Auto Answer,
Auto Dial, FREE Software
$199
NOVATION
1200 BAUD SPECIAL $529
1200 BAUD UPGRADE $339

DISK DRIVES
Concorde CIII $189
Rana Elite I $229
Rana Elite II $369
Rana Elite III $475

TAXAN
MODEL
115 12" Amber Hi-Res $129
210 12" Composite-RGB $289
400 12" RGB Med-Res $299
410 12" RGB Hi-Res $349
415 12" RGB Super Hi-Res $399

12629 N. Tatum Blvd.
Suite 212
Phoenix, AZ 85032

Call: 602-957-3619

$2 PHONE REBATE
WITH ANY ORDER

SHIPPING CHARGES
0-100 $ 5
101-200 $ 8
201-300 $10
301- up $15
All prices are for cash or check—
Visa/Mastercard add 3%

Not very professional. Luckily, Mailmerge will also accept an overkill method of data separation: enclosing all data in quotes and separating them with commas. This seems like it would be more work, but actually it is the uniformity of this approach that makes this method work. And dBASE II does all the work for us.

Now let's try the conventional method of creating text files from dBASE II database files. While in dBASE II use the command:

```
. USE DATA
. COPY TO datatext DELIMITED WITH"
```

If we now examine the file DATATEXT.TXT (the .TXT extension is automatically assigned by dBASE II) using Wordstar or a suitable command like CP/M's TYPE, we'll see that all data is enclosed in quotes but with all the trailing blacks carried right along like this:

```
"Mr. ","John H.       ","Stein        ","Tate and Sons       ",
"24 Lake Avenue    ","Chicago      ","IL   ","60606   ","Mrs.",
```

Well, that's no good. Let's try the other way of creating text files from dBASE II files:

```
. USE DATA
. COPY TO datatext DELIMITED WITH,
```

Using this method, all data will be separated by commas and the trailing blanks will be trimmed from each piece of data like this:

```
Mr.,John H.,Stein,Tate and Sons,24 Lake Avenue,Chicago,IL.60606,
Mrs.,Joanne T.,Houseman,Melt,Inc.,970 Tulips Street,New York,NY,
10019,Ms.,Toni J.,Deltawashington,Lewis, Jones, and Co.,11678
Riverside Dr.,Burbank,CA,91356
```

Unfortunately, embedded commas (like in Melt, Inc.) will cause incorrect data tracking as we discussed earlier. Obviously this won't work with all data types either.

Fortunately, there is a trickier way to create text files with dBASE II which allows the full power of the program to be running at the same time. If the command SET ALTERNATE TO datatext.txt is used followed by SET ALTERNATE ON, everything shown on the screen will be sent to the file datatext.txt. So, we can write a short dBASE II program stripping away the trailing blainks from the data using the TRIM funciton and insert the proper punctuation so Mailmerge gets just what it wants. And, as an added bonus, since the full power of dBASE II is available, we can "filter" the usable data for our form letters much more efficiently than we could by simply using the COPY TO command line.

For our example, the following program could be written in dBASE II:

```
USE DATA
SET TALK OFF
SET RAW ON
SET ALTERNATE TO datatext.txt
SET ALTERNATE ON
DO WHILE .NOT. EOF
? '"' , TRIM(MRMRS)      , '"' , ',' ,
? '"' , TRIM(FIRSTNAME)  , '"' , ',' ,
? '"' , TRIM(LASTNAME)   , '"' , ',' ,
? '"' , TRIM(COMPANY)    , '"' , ',' ,
? '"' , TRIM(STREET)     , '"' , ',' ,
? '"' , TRIM(CITY)       , '"' , ',' ,
? '"' , TRIM(STATE)      , '"' , ',' ,
? '"' , TRIM(ZIP)        , '"' , ',' ,
SKIP
ENDDO
SET ALTERNATE OFF
```

The SET TALK OFF command stops dBASE II from sending comments to the screen as it makes calculations or skips to the next record. SET RAW ON eliminates single spaces between fields which when defaulted to OFF is the normal way dBASE II will LIST or DISPLAY data. Keep in mind that everything sent to the screen will appear in the file datatext.txt.

Using the above dBASE II program on our sample file data.dbf will yield the following result automatically given the filename datatext.txt by dBASE II:

```
"Mr."
"John H.",
"Stein",
"Tate and Sons",
"24 Lake Avenue",
"Chicago",
"IL",
"60606",
"Mrs.",
"Joanne T.",
"Houseman",
"Melt, Inc.",
"970 Tulips Street",
"New York",
"NY",
"10019",
"Ms.",
"Toni J.",
"Deltawashington",
"Lewis, Jones and Co",
"11678 Riverside Dr.",
"Burbank",
"CA",
"91356",
```

We can see that this format adheres to all of Mailmerge's rules. Each piece of data is separated by commas, trailing blanks trimmed and enclosed in quotes to protect embedded commas within data fields.

An extra added bonus is that our original dBASE II command file used to create the datatext.txt file is casily modified for use with other dBASE II .DBF files by changing the field namcs only via Wordstar or dBASE II's Modify Command. The rest remains intact.

If you do any work at all with these programs and form letters, you'll find this method unbeatable.

MICRO

STEPPER

A Step-Trace facility with a handy twist.

by Chester H. Page
Silver Spring, Maryland

Have you ever tried to debug a machine-language program by step-tracing it and found that you were looking at a lot of instructions that you had not written? This can be slow and annoying.

The reason for this phenomenon is obvious — all JSR calls to MONITOR routines get traced through step-by-step and this is distracting. I decided that I wanted a step-trace program that would display only my instructions, i.e., any call to a monitor subroutine would be handled as a single step operation, no matter how many monitor steps were actually involved.

For background, I studied Peterson's Step-Trace in MICRO on the Apple, Volume 2. That program mimics the Step-Trace facility that was in the old monitor ROM before the days of AutoStart. He added a few features such as interrupted trace. In addition to the nuisance of dissecting all monitor subroutines used in your program, this Step-Trace program has a problem with COUT — it hangs up. My modification avoids this problem because COUT is not dissected. In fact, I can step-trace a program which calls for printing a word on the screen, then activating the printer and repeating the print operation on paper! To do this requires one special procedure: instead of changing output hooks at $36/37 (which would cause a hang-up) the hooks are changed directly in DOS at $AA53/AA54.

For a final touch, my stepper program is relocatable and can be BRUN at any location that avoids the program to be debugged.

The step-tracing displays of instructions and user registers are placed on screen lines 0 to 19; user COUT output appears in lines 22 and 23. Lines 20 and 21 maintain a gap for better appearance. The right-hand end of the gap lines is used for special storage, so exhibits a peculiar combination of characters. The reason for this unconventional storage location is to avoid possible interference with user programs which may use any available zero-page space

With these tools...

S-C Macro Assembler – Combined editor/assembler includes 29 commands and 20 directives, with macros, conditional assembly, global replace, edit, and more. Well-known for ease-of-use and powerful features. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependability, and user-friendliness. Blends power, simplicity, and performance to provide the optimum capabilities to both beginning and professional programmers. With 100-page manual and reference card, $92.50

Cross Assembler Modules – Owners of the S-C Macro Assembler may add the ability to develop programs for other systems. We have modules for most of the popular chips, at very reasonable prices

6800/01/02	$32.50
6805	$32.50
6809	$32.50
68000	$50.00
Z-80	$32.50
PDP-11	$50.00
8048	$32.50
8051	$32.50
8080/85	$32.50
1802/04/05	$32.50

All of the cross assemblers retain the full power of the S-C Macro Assembler. You can develop programs for burning into EPROMs, transfer through a data-link, or direct execution by some of the plug-in processor cards now on the market.

Apple Assembly Line – Monthly newsletter for assembly language programmers, beginner or advanced. Tutorial articles; advanced techniques; handy utility programs; commented listings of code in DOS and Apple ROMs; reviews of relevant new books, hardware, software; and more! $18 per year (add $3 for first class postage in USA, Canada, Mexico; add $13 postage for other countries).

S-C Software Corporation
2331 Gus Thomasson, Suite 125
Dallas, Texas 75228
(214) 324-2050

and may use the empty spaces in DOS for data storage. This special storage area is for (1) the split-screen window data, (2) storage of output-hook data and (3) storage of the XQT (execute instruction) area. The old monitor ROM uses $3C/44 for XQT, but $3E/3F gets overwritten by GETNUM. For this reason, the XQT area had to be re-initialized before each user step. I preferred to avoid this repetition because I use a lengthy initialization routine for relocatability. The problem is in the two jump instructions that follow the user command which was copied to XQT. These two jumps are for the normal no-branch return and the special return when relative-jump branches are taken. Since these jumps are to routines within the stepper program, they must be inserted by a code-locating routine. I preferred running this just once at the beginning of the program.

STEPPER is conventional except for one major departure. After a user instruction is copied to XQT, it is examined to see if it calls an address in the monitor. If so, the instruction is executed directly instead of being simulated and stepped through. In the case of an indirect jump, the indicated jump address is examined for location. If it is in the monitor, it is displayed and the instruction executed.

To use STEPPER, load the program to be debugged (at $nnnn), enter HOME, then BRUN STEPPER, AX at any convenient location. Your program can be stepped through by entering nnnnS, followed by entering an S for each successive step. Alternatively, nnnnT will produce a continuous trace of successive steps which can be interrupted by pressing any key and restarted with T, or shifted to single steps with S.

STEPPER can be tested on STEPPER DEMO. STEPPER DEMO includes subroutines, direct jumps and indirect jumps, both within itself and to the monitor and hook changing. To run, BLOAD STEPPER DEMO, enter HOME, BRUN STEPPER and enter 300T. The word "TEST" should be printed on the bottom screen line, then the printer activated and "TEST" printed on paper.

Listing 1 Note: Mr. Page uses the S-C Macro Assembler, published by S-C Software Corporation

```
*****************************
*         STEPPER           *
*        CHET PAGE          *
*****************************
                .OR $7000
                .TA $8000
*               .TF STEPPER
0022-           W EQU $22
002F-           LENGTH EQU $2F
0033-           PROMPT EQU $33
0034-           YSAV EQU $34
003A-           PCL EQU $3A
003B-           PCH EQU $3B
003C-           INDPTR EQU $3C
0048-           STATUS EQU $48
0100-           STACK EQU $100
066E-           OUTPRT EQU $66E
06EF-           XQT EQU $6EF
0670-           UW EQU $670
0674-           TW EQU $674
AA53-           HOOK EQU $AA53
C000-           KBRD EQU $C000
F882-           INSDS1 EQU $F882
F8D0-           INSTDSP EQU $F8D0
F948-           PRBLNK EQU $F948
F954-           PCADJ2 EQU $F954
F956-           PCADJ3 EQU $F956
FAD7-           REGDSP EQU $FAD7
FC22-           VTAB EQU $FC22
FD67-           GETLNZ EQU $FD67
FDDA-           PRBYTE EQU $FDDA
FDED-           COUT EQU $FDED
FE00-           BL1 EQU $FE00
FE75-           A1PC EQU $FE75
FF3A-           BELL EQU $FF3A
FF3F-           RESTORE EQU $FF3F
FF4A-           SAVE EQU $FF4A
FFA7-           GETNUM EQU $FFA7
FF58-           RTRN EQU $FF58
FFBE-           TOSUB EQU $FFBE
FFC5-           ZMOD0 EQU $FFC5
FFC7-           ZMODE EQU $FFC7
FFCC-           CHRTBL EQU $FFCC
*---------------------------
*INITIALIZE WINDOW AREAS
*---------------------------
7000- A9 00             LDA #0
7002- 85 22             STA W
7004- 8D 72 06          STA UW+2
7007- A9 14             LDA #$14
7009- 85 23             STA W+1
700B- A9 13             LDA #$13
700D- 85 25             STA W+3
700F- A9 16             LDA #$16
7011- 8D 70 06          STA UW
7014- A9 18             LDA #$18
7016- 8D 71 06          STA UW+1
7019- A9 17             LDA #$17
701B- 8D 73 06          STA UW+3
*---------------------------
701E- AD 53 AA          LDA HOOK
7021- 8D 6E 06          STA OUTPRT
7024- AD 54 AA          LDA HOOK+1
7027- 8D 6F 06          STA OUTPRT+1
```

```
*-------------------
* INITIALIZE XEQ RETURNS
*-------------------
702A- 20 58 FF        JSR RTRN
702D- B8              CLV
702E- 50 07           BVC NB
7030- 20 4A FF NBRN   JSR SAVE
7033- 38              SEC
7034- B8              CLV
*RELAY TO PCN3 LOCATE NBRN
* AND INSERT JUMP
7035- 50 6D           BVC PCN3A1
7037- BA       NB     TSX
7038- CA              DEX
7039- 18              CLC
703A- BD 00 01        LDA STACK,X
703D- 69 04           ADC #4
703F- 8D F3 06        STA XQT+4
7042- E8              INX
7043- BD 00 01        LDA STACK,X
7046- 69 00           ADC #0
7048- 8D F4 06        STA XQT+5
704B- A9 4C           LDA #$4C
704D- 8D F2 06        STA XQT+3
7050- 8D F5 06        STA XQT+6
7053- 20 58 FF        JSR RTRN
7056- B8              CLV
7057- 50 0F           BVC BR
7059- 18       BRAN   CLC
705A- A0 01           LDY #1
705C- B1 3A           LDA (PCL),Y
705E- 20 56 F9        JSR PCADJ3
7061- 85 3A           STA PCL
7063- 98              TYA
7064- 38              SEC
7065- B8              CLV
*RELAY TO PCN2 LOCATE BRAN
* AND INSERT JUMP
7066- 50 3E           BVC PCN2A1
7068- BA       BR     TSX
7069- CA              DEX
706A- 18              CLC
706B- BD 00 01        LDA STACK,X
706E- 69 04           ADC #4
7070- 8D F6 06        STA XQT+7
7073- E8              INX
7074- BD 00 01        LDA STACK,X
7077- 69 00           ADC #0
7079- 8D F7 06        STA XQT+8
*-------------------------
707C- D8       STRT   CLD
707D- 20 3A FF        JSR BELL
7080- A9 2A    CONT   LDA #$2A
7082- 85 33           STA PROMPT
7084- 20 67 FD        JSR GETLNZ
7087- 20 C7 FF        JSR ZMODE
708A- 20 A7 FF NXTI   JSR GETNUM
708D- 84 34           STY YSAV
*IS IT STEP? IS IT TRACE?
* IS IT <CR> ?
708F- C9 EC           CMP #$EC
7091- F0 1E           BEQ ENT
7093- C9 ED           CMP #$ED
7095- F0 13           BEQ TRACE
7097- C9 C6           CMP #$C6
7099- D0 1C           BNE MCMD
709B- 20 C5 FF        JSR ZMOD0
709E- 20 00 FE        JSR BL1
70A1- B8              CLV
70A2- 50 DC           BVC CONT

*RELAYS
70A4- 50 58    PCN3A1 BVC  PCN3A2
70A6- 50 58    PCN2A1 BVC PCN2A2
70A8- 50 D2    STRT1  BVC STRT
70AA- AD 00 C0 TRACE  LDA KBRD
*STOP ON ANY KEY
* WAIT FOR NEXT, REPEAT STEP
70AD- 30 15           BMI AGIN
70AF- C6 34           DEC YSAV
70B1- 20 C7 FF ENT    JSR ZMODE
70B4- B8              CLV
70B5- 50 12           BVC STEP
*TRY MONITOR COMMANDS
*SEARCH MON CHARS
70B7- A0 17    MCMD   LDY #$17
70B9- 88       CHRS   DEY
70BA- 30 C0           BMI STRT
*CMP WITH TABLE, TRY AGAIN
* FOUND, PROCEED
70BC- D9 CC FF        CMP CHRTBL,Y
70BF- D0 F8           BNE CHRS
70C1- 20 BE FF        JSR TOSUB
70C4- A4 34    AGIN   LDY YSAV
70C6- B8              CLV
*GET NEXT COMMAND, ADR TO PC
* DISPLAY INSTRUCTION
* RESET OUTPUT PORT
70C7- 50 C1           BVC NXTI
70C9- 20 75 FE STEP   JSR A1PC
70CC- 20 D0 F8        JSR INSTDSP
70CF- AD 6E 06        LDA OUTPRT
70D2- 8D 53 AA        STA HOOK
70D5- AD 6F 06        LDA OUTPRT+1
70D8- 8D 54 AA        STA HOOK+1
70DB- A2 02           LDX #2
*NOP'S TO XEQ AREA
70DD- A9 EA    XQIN   LDA #$EA
70DF- 9D EF 06        STA XQT,X
70E2- CA              DEX
70E3- D0 F8           BNE XQIN
*-------------------
* COPY USER COMMAND
*-------------------
70E5- A2 00           LDX #0
70E7- A1 3A           LDA (PCL,X)
70E9- D0 17           BNE NOTBRK
*SET SCREEN OUTPUT
70EB- A9 F0           LDA #$F0
70ED- 8D 53 AA        STA HOOK
70F0- A9 FD           LDA #$FD
70F2- 8D 54 AA        STA HOOK+1
70F5- 20 82 F8        JSR INSDS1
70F8- 20 D7 FA        JSR REGDSP
70FB- B8              CLV
*RELAY
70FC- 50 AA           BVC STRT1
70FE- 50 7B    PCN3A2 BVC PCN3A3
7100- 50 7B    PCN2A2 BVC PCN2A3
*-------------------
* EXAMINE OPCODE
*-------------------
7102- A4 2F    NOTBRK LDY LENGTH
7104- C9 20           CMP #$20
7106- D0 0A           BNE TRYJMP
*IT IS JSR
*IS IT IN MONITOR?
*YES, SO EXECUTE, NO, SIMULATE
7108- B1 3A           LDA (PCL),Y
710A- C9 F8           CMP #$F8
710C- B0 71           BCS EX1
```

```
710E- 90 73           BCC SJSR1
7110- 50 B2    AGIN2  BVC AGIN
7112- C9 4C    TRYJMP CMP #$4C
7114- D0 08           BNE TRYIND
*IT IS JMP
*IS IT TO MONITOR?
*YES, EXECUTE  NO, SIMULATE
7116- B1 3A           LDA (PCL),Y
7118- C9 F8           CMP #$F8
711A- B0 69           BCS EX2
711C- 90 69           BCC SJMP
711E- C9 6C    TRYIND CMP #$6C
7120- D0 67           BNE TRYRTS
*INDIRECT JUMP
7122- B1 3A           LDA (PCL),Y
7124- 85 3D           STA INDPTR+1
7126- 88              DEY
7127- B1 3A           LDA (PCL),Y
7129- 85 3C           STA INDPTR
712B- B1 3C           LDA (INDPTR),Y
*IS IT TO MONITOR? SIMULATE
712D- C9 F8           CMP #$F8
712F- 90 45           BCC SIND1
*-------------------------------
* DISPLAY MONITOR ADDRESS
*-------------------------------
7131- AD 53 AA        LDA HOOK
7134- 8D 6E 06        STA OUTPRT
7137- AD 54 AA        LDA HOOK+1
713A- 8D 6F 06        STA OUTPRT+1
713D- A9 F0           LDA #$F0
713F- 8D 53 AA        STA HOOK
7142- A9 FD           LDA #$FD
7144- 8D 54 AA        STA HOOK+1
7147- A9 8D           LDA #$8D
7149- 20 ED FD        JSR COUT
714C- 20 48 F9        JSR PRBLNK
714F- 20 48 F9        JSR PRBLNK
7152- 20 48 F9        JSR PRBLNK
7155- A9 A4           LDA #$A4
7157- 20 ED FD        JSR COUT
715A- B1 3C           LDA (INDPTR),Y
715C- 20 DA FD        JSR PRBYTE
715F- 88              DEY
7160- B1 3C           LDA (INDPTR),Y
7162- 20 DA FD        JSR PRBYTE
7165- AD 6E 06        LDA OUTPRT
7168- 8D 53 AA        STA HOOK
716B- AD 6F 06        LDA OUTPRT+1
716E- 8D 54 AA        STA HOOK+1
*-------------------------------
7171- A4 2F           LDY LENGTH
7173- B8              CLV
7174- 50 43           BVC USER1
7176- A4 2F    SIND1  LDY LENGTH
7178- 38              SEC
7179- B0 4E           BCS SIND
*RELAYS
717B- 50 34    PCN3A3 BVC PCN3
717D- 50 30    PCN2A3 BVC PCN2
717F- B0 3A    EX1    BCS USER2
7181- 50 8D    AGIN1  BVC AGIN2
7183- 90 38    SJSR1  BCC SJSR
7185- B0 32    EX2    BCS USER1
7187- 90 3F    SJMP1  BCC SJMP
7189- C9 60    TRYRTS CMP #$60
718B- F0 1E           BEQ SRTS
718D- C9 40           CMP #$40
718F- F0 16           BEQ SRTI
7191- 29 1F           AND #$1F
7193- 49 14           EOR #$14
7195- C9 04           CMP #4
*SET UP BRANCH, RETURN TO BRAN
7197- F0 02           BEQ XQ2
7199- B1 3A    XQ1    LDA (PCL),Y
719B- 99 EF 06 XQ2    STA XQT,Y
719E- 88              DEY
719F- 10 F8           BPL XQ1
*RESTORE USER REGISTERS, EXEC USER CMD
71A1- 20 3F FF        JSR RESTORE
71A4- 4C EF 06        JMP XQT
*SIMULATE RTI
71A7- 18       SRTI   CLC
71A8- 68              PLA
71A9- 85 48           STA STATUS
*SIMULATE RTS
71AB- 68       SRTS   PLA
71AC- 85 3A           STA PCL
71AE- 68              PLA
71AF- 85 3B    PCN2   STA PCH
71B1- 20 54 F9 PCN3   JSR PCADJ2
71B4- 84 3B           STY PCH
71B6- 18              CLC
71B7- 90 18           BCC NEWP
*RELAYS
71B9- 50 43    USER1  BVC USER4
71BB- 50 36    USER2  BVC USER3
*SIMULATE JSR
*SIMULATE RETURN TO STACK
71BD- 18       SJSR   CLC
71BE- 20 54 F9        JSR PCADJ2
71C1- AA              TAX
71C2- 98              TYA
71C3- 48              PHA
71C4- 8A              TXA
71C5- 48              PHA
71C6- A0 02           LDY #2
*SIMULATE JUMP
71C8- 18       SJMP   CLC
*SIMULATE INDIRECT JUMP
71C9- B1 3A    SIND   LDA (PCL),Y
71CB- AA              TAX
71CC- 88              DEY
71CD- B1 3A           LDA (PCL),Y
71CF- 86 3B           STX PCH
71D1- 85 3A    NEWP   STA PCL
71D3- B0 F3           BCS SJMP
71D5- AD 53 AA NEWP2  LDA HOOK
71D8- 8D 6E 06        STA OUTPRT
71DB- AD 54 AA        LDA HOOK+1
71DE- 8D 6F 06        STA OUTPRT+1
71E1- A9 F0           LDA #$F0
71E3- 8D 53 AA        STA HOOK
71E6- A9 FD           LDA #$FD
71E8- 8D 54 AA        STA HOOK+1
71EB- 20 D7 FA        JSR REGDSP
71EE- B8              CLV
*RELAY TO AGIN
71EF- 50 90           BVC AGIN1
71F1- 50 E2    NEWP1  BVC NEWP2
*-------------------------------
*EXECUTE MONITOR ROUTINE
*-------------------------------
*IF MONITOR SUBROUTINE SET UP RETURN
*AND CONVERT JSR TO JMP
71F3- 18       USER3  CLC
71F4- 20 54 F9        JSR PCADJ2
71F7- AA              TAX
71F8- 98              TYA
71F9- 48              PHA
```

```
71FA- 8A              TXA
71FB- 48              PHA
71FC- A0 02           LDY #2
*EXECUTE MONITOR ROUTINE
71FE- A9 60    USER4  LDA #$60
7200- 8D F2 06        STA XQT+3
7203- B1 3A    .1     LDA (PCL),Y
7205- 99 EF 06        STA XQT,Y
7208- 88              DEY
7209- 10 F8           BPL .1
*IF JSR, CONVERT TO JMP
720B- C9 20           CMP #$20
720D- D0 06           BNE .2
720F- A9 4C           LDA #$4C
7211- C8              INY
7212- 99 EF 06        STA XQT,Y
7215- A2 03    .2     LDX #3
*SAVE TRACE WINDOW DATA
7217- B5 22    SVT    LDA W,X
7219- 9D 74 06        STA TW,X
721C- CA              DEX
721D- 10 F8           BPL SVT
721F- A2 03           LDX #3
*LOAD USER-WINDOW DATA
7221- BD 70 06 LDU    LDA UW,X
7224- 95 22           STA W,X
7226- CA              DEX
7227- 10 F8           BPL LDU
7229- 20 22 FC        JSR VTAB
722C- 20 3F FF        JSR RESTORE
722F- 20 EF 06        JSR XQT
7232- 20 4A FF        JSR SAVE
7235- A2 03           LDX #3
*SAVE USER-WINDOW DATA
7237- B5 22    SVU    LDA W,X
7239- 9D 70 06        STA UW,X
723C- CA              DEX
723D- 10 F8           BPL SVU
723F- A2 03           LDX #3
*LOAD TRACE-WINDOW DATA
7241- BD 74 06 LDT    LDA TW,X
7244- 95 22           STA W,X
7246- CA              DEX
7247- 10 F8           BPL LDT
7249- A9 4C           LDA #$4C
724B- 8D F2 06        STA XQT+3
*RECOVER RETURN ADDRESS
724E- 68              PLA
724F- 85 3A           STA PCL
7251- 68              PLA
7252- 85 3B           STA PCH
7254- E6 3A           INC PCL
7256- D0 02           BNE .2
7258- E6 3B           INC PCH
725A- A2 00    .2     LDX #0
725C- B8              CLV
*RELAY TO NEWP2
725D- 50 92           BVC NEWP1
```

For those who aren't using an S-C Assembler, to the right are definitions of those commands which are unique to this assembler. The following are from the 'S-C Macro Assembler' by Bob Sander-Cederlof, copyright 1982, S-C Software Corporation.

Listing 2

```
                      1000 ******************
                      1010 *  STEPPER DEMO  *
                      1020 ******************
                      1030         .OR $300
                      1040         .TA $800
                      1050         .TF STEPPER DEMO
FDED-                 1060   COUT .EQ $FDED
0006-                 1070    PTR .EQ $6
0300- 20 11 03 1080        JSR TEXT
0303- A9 00    1090        LDA #0
0305- 8D 53 AA 1100        STA $AA53
0308- A9 C1    1110        LDA #$C1
030A- 8D 54 AA 1120        STA $AA54
030D- 20 11 03 1130        JSR TEXT
0310- 00       1140        BRK
0311- A9 8D    1150 TEXT   LDA #$8D
0313- 20 ED FD 1160        JSR COUT
0316- A9 D4    1170        LDA #$D4
0318- 20 44 03 1180        JSR T1
031B- A9 C5    1190        LDA #$C5
031D- 20 47 03 1200        JSR T2
0320- A9 D3    1210        LDA #$D3
0322- 20 4F 03 1220        JSR T5
0325- 20 58 FF 1230        JSR $FF58
0328- 20 54 03 1240        JSR T7
032B- A2 41    1250        LDX #WORD
032D- A0 03    1260        LDY /WORD
032F- 86 06    1270        STX PTR
0331- 84 07    1280        STY PTR+1
0333- A0 00    1290        LDY #0
0335- B1 06    1300 PRINT  LDA (PTR),Y
0337- F0 07    1310        BEQ DONE
0339- 20 ED FD 1320        JSR COUT
033C- C8       1330        INY
033D- 4C 35 03 1340        JMP PRINT
0340- 60       1350 DONE   RTS
0341- D4 8D 00 1360 WORD   .HS D48D00
0344- 4C ED FD 1370 T1     JMP COUT
0347- 6C 4A 03 1380 T2     JMP (T3)
034A- 4C       1390 T3     .DA #T4
034B- 03       1400        .DA /T4
034C- 4C 3C FF 1410 T4     JMP $FF3C
034F- 6C 52 03 1420 T5     JMP (T6)
0352- ED FD    1430 T6     .HS EDFD
0354- 6C 57 03 1440 T7     JMP (T8)
0357- 58 FF    1450 T8     .HS 58FF
```

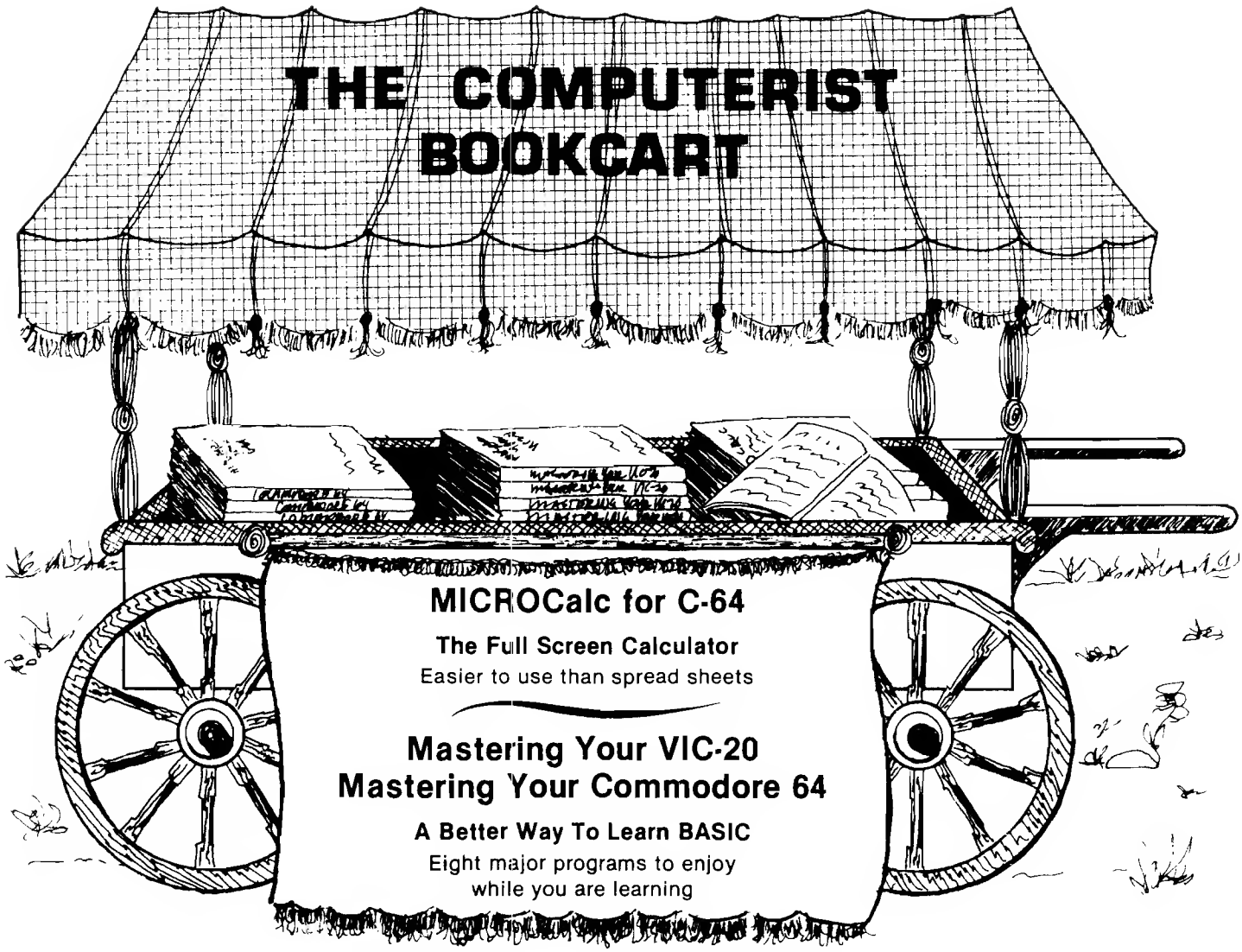
.DA DAta
'Creates constants or variables in your program...The value of the expression, as one or two bytes, is stored at the current location. If a label is present, it is defined as the address where the first byte of data is stored.'

.HS Hex String
'Converts a string of hex digits (hhh...h) to binary, two digits per byte, and stores them starting at the current location. If a label is present, it is defined as the address where the first byte is stored.'

.TF Target File
'Causes the object code generated to be stored on a binary file, rather than in memory.'

.TA Target Address
'Sets the target address at which the object-code will be stored during assembly.'

.OR Origin
'Sets the program origin and the target address to the value of the expression. Program origin is the address at which the object program will be executed.'



Mastering Your VIC-20
Mastering Your Commodore 64

The 8 programs, "run-ready" on disk (C-64) or tape (VIC-20) and explained in the 160-192 page book, each demonstrate important concepts of BASIC while providing useful, enjoyable software. Programs include:

- Player — compose songs from your keyboard, save, load and edit for perfect music
- MicroCalc — display calculation program that make even complex operations easy
- Master — a one or two person guessing game
- Clock — character graphics for a digital clock

VIC-20 with tape & book ..... **just $19.95**
C-64 with disk & book (avail. Sept.) ..... **just $19.95**

Look for us at the
International Software Show
Toronto, September 20-23

MICROCalc for C-64

This on-screen calculator comes with diskette and 48-page manual offering a wide variety of useful screens, and a great way to learn BASIC expressions if you don't already know them.

- Unlimited calculation length & complexity
- Screens can be linked and saved on disk/cassette
- Build a library of customized screens
- Provide formatted printer output

Diskette & 48-page manual ..... **just $29.95**

For the Freshest Books, Buy Direct!

- No prehandled books with bent corners
- Books come direct to your door
- No time wasted searching store to store
- 24 hours from order receipt to shipment
- No shipping/handling charges
- No sales tax (except 5% MA res.)
- Check, MO, VISA/MC accepted (prepaid only)

The Computerist Bookcart
P.O. Box 6502, Chelmsford, MA 01824
For faster service, phone: 617/256 – 3649.

Time-Series Forecasting

by Brian Flynn
Vienna, Virginia

A program to predict the future.

Requirements: Apple II, Commodore 64, Atari, or CoCo with Flex

Prophets and pundits since time immemorial have tried to peer into that murky and mysterious region of shifting shadows and dancing dreams called the future. These seers and soothsayers have plied their fortune-telling trade using every sort of contrivance imaginable: everything from shooting stars and playing cards to chicken entrails and crystal balls.

Today you can join this elite group of mystics and Merlins by using the time-series forecasting program presented here. It's written in a 'generic BASIC' and runs nicely on a machine with minimum memory. And while the forecasts of yesteryear were often as illusory as a maintenance-free automobile or a meek and mild Klingon, the microcomputer projections can have a sound scientific basis.

After first explaining what a "time series" is, we'll show you how to use the microcomputer program to predict future interest rates. We'll describe the program's forecasting techniques

1. Least-Squares Trend
2. Semi-Averages
3. Percent Changes
4. First Differences, and
5. Past Averages

and give you hints on when to use each.

What's a Time Series?

A time series is a group of observations on a variable, in chronological order, at a set frequency. Monthly sales, weekly income, annual gross national product and the number of Americans flying to London every August are examples.

The adjective "time" means that our observations are tallied at equal calendar intervals. And the noun "series" means that we have more than one data point. Hence, a "time series" records a variable's past and time-series forecasting projects its future using historical observations.

As the sage says, the past is prologue, or so we hope.

Real-World Example: Whither Interest Rates?

The future level of interest rates in the economy concerns most of us, borrowers and lenders alike. If we're thinking about financing a new house or car, for example, and if interest rates are falling, then delay will save us dollars. But we're better off buying now if interest rates are rising.

Suppose the dogs of debt are muzzled for a change, however, and that we strut proudly to the teller's window to lend instead of to borrow. Cash in hand and nirvana in mind, we decide to plop down $5K for a money-market certificate maturing somewhere between 3 months and 2 1/2 years. If interest rates are headed downward, locking in a relatively high rate now for as long as possible (2 1/2 years) is sound strategy. With rising interest rates, on the other hand, we're better off with a shorter maturity.

There's only one catch to this nice "buy-low sell-high" kind of advice. How do we know if future interest rates will rise or fall? To try and find out,

let's use the time-series forecasting routine. Our first step is to gather historical data, with our bountiful harvest shown in Table 1. This sequence of numbers qualifies as a time series: the observations are in chronological order (August 1982 to April 1984) at a set frequency (monthly).

Table 1

Interest Rates on
3-Month Treasury Bills

Year:	Month	Percent
1982:	8	8.68
	9	7.92
	10	7.71
	11	8.07
	12	7.94
1983:	1	7.86
	2	8.11
	3	8.35
	4	8.21
	5	8.19
	6	8.79
	7	9.08
	8	9.34
	9	9.00
	10	8.64
	11	8.76
	12	9.00
1984:	1	8.90
	2	9.09
	3	9.52
	4	9.69

After keying our figures into the computer, with the program prompting us at every turn, the microcomputer indicates that future interest rates can be predicted using any of these extrapolation techniques: (1) Least-Squares Trend, (2) Semi-Averages, (3) Percent Changes, (4) First Differences, and (5) Past Averages.

We can predict as far into the future as we care or dare, using one method after another, until all reasonable alternatives are exhausted.

We now explain each technique in detail.

Least-Squares Trend

In Least-Squares Trend the microcomputer forecasts by extrapolating an historically fitted regression line into the future. As Figure 1 shows, the technique satisfies an urge that almost all of us have had: to draw a line through a plot of points to best reflect the apparent trend.

The microcomputer estimates the line using a statistical technique called "ordinary least squares". Our dependent variable, interest rates, is regressed on a lone explanatory variable, "time". Observations on the latter are generated internally by the program, with August 1982 corresponding to 1 (the first time period), September 1982 corresponding to 2 (the second time period), and so on. April 1984 corresponds to 21 since we have 21 months worth of data.

Technically, the microcomputer estimates our linear equation so that the sum of squared deviations of our observations from the line is as small as possible; hence the term "least squares".

Taking a practical example, let's forecast interest rates in January 1985. We ask the microcomputer to project nine months ahead, from May 1984. It responds with the forecasts of Table 2.

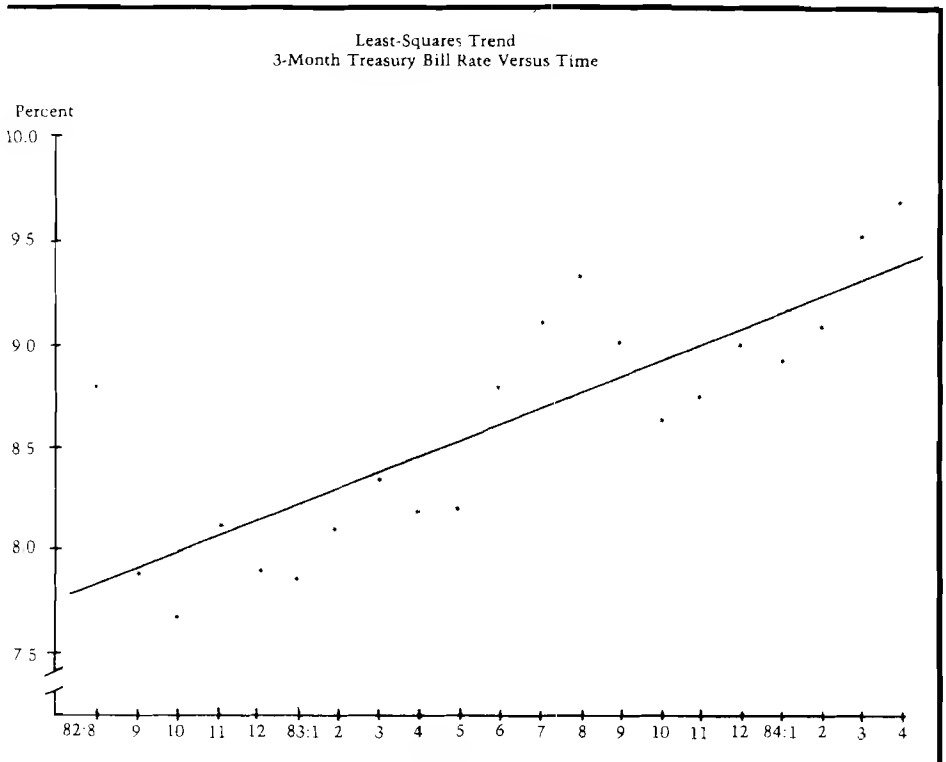


Figure 1

```
================================
        FORECASTS
================================
METHOD: LEAST-SQUARES TREND

PERIOD (  22  )=  9.46
PERIOD (  23  )=  9.53
PERIOD (  24  )=  9.61
PERIOD (  25  )=  9.69
PERIOD (  26  )=  9.77
PERIOD (  27  )=  9.84
PERIOD (  28  )=  9.92
PERIOD (  29  )=  10
PERIOD (  30  )=  10.07
================================

================================
        FORECASTS
================================
METHOD: SEMI-AVERAGES

PERIOD (  22  )=  9.62
PERIOD (  23  )=  9.72
PERIOD (  24  )=  9.81
PERIOD (  25  )=  9.9
PERIOD (  26  )=  9.99
PERIOD (  27  )=  10.08
PERIOD (  28  )=  10.18
PERIOD (  29  )=  10.27
PERIOD (  30  )=  10.36
================================

================================
        FORECASTS
================================
METHOD: PERCENT CHANGES

PERIOD (  22  )=  9.86
PERIOD (  23  )=  10.03
PERIOD (  24  )=  10.21
PERIOD (  25  )=  10.4
PERIOD (  26  )=  10.58
PERIOD (  27  )=  10.77
PERIOD (  28  )=  10.96
PERIOD (  29  )=  11.16
PERIOD (  30  )=  11.36
================================
```

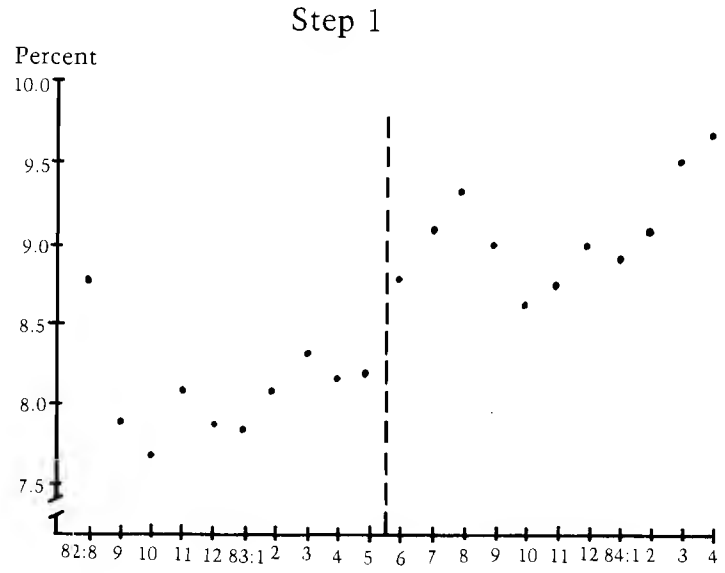
```
==============================
        FORECASTS
==============================
METHOD: FIRST DIFFERENCES

PERIOD (  22  )=  9.86
PERIOD (  23  )=  1Ø.Ø3
PERIOD (  24  )=  1Ø.2
PERIOD (  25  )=  1Ø.37
PERIOD (  26  )=  1Ø.54
PERIOD (  27  )=  1Ø.71
PERIOD (  28  )=  1Ø.88
PERIOD (  29  )=  11.Ø5
PERIOD (  3Ø  )=  11.22
==============================

==============================
        FORECASTS
==============================
METHOD: PAST AVERAGES OF  2Ø

PERIOD (  22  )=  9.75
PERIOD (  23  )=  9.81
PERIOD (  24  )=  9.87
PERIOD (  25  )=  9.93
PERIOD (  26  )=  9.99
PERIOD (  27  )=  1Ø.Ø5
PERIOD (  28  )=  1Ø.11
PERIOD (  29  )=  1Ø.18
PERIOD (  3Ø  )=  1Ø.24
==============================

==============================
        FORECASTS
==============================
METHOD: PAST AVERAGES OF  2Ø

PERIOD (  22  )=  9.74
PERIOD (  23  )=  9.79
PERIOD (  24  )=  9.84
PERIOD (  25  )=  9.89
PERIOD (  26  )=  9.94
PERIOD (  27  )=  9.99
PERIOD (  28  )=  1Ø.Ø4
PERIOD (  29  )=  1Ø.Ø9
PERIOD (  3Ø  )=  1Ø.14
==============================
```

A big advantage of Least-Squares Trend over traditional regression routines is that we know future values of the explanatory variable ("time") with absolute certainty. With the latter technique we don't. Using the rate of inflation instead of "time" to predict interest rates, for example, leaves us the problem of estimating the price level in January 1985 before we can run our model.

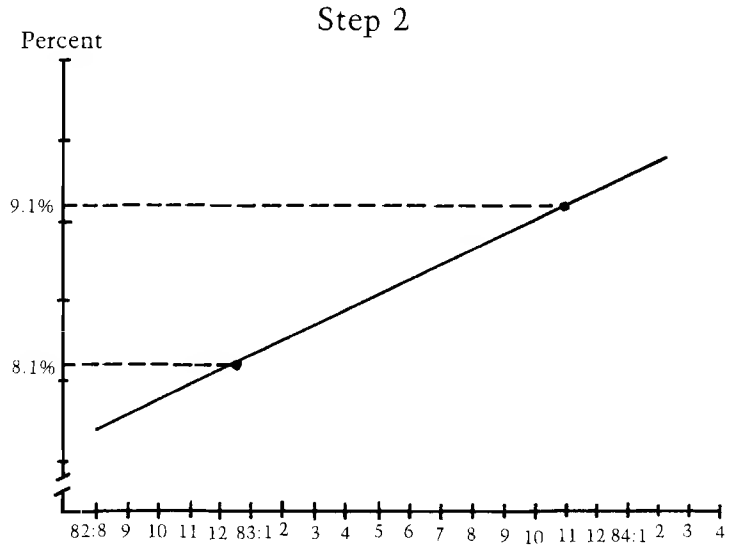
Semi-Averages

The method of Semi-Averages is somewhat akin to Least-Squares Trend. Namely, the microcomputer divides our time series into two roughly equal parts and computes the mean of each. Then it fits a line through the two points, with values on the line representing our forecasts. Figure 2 details the process.

Method of Semi-Averages



In the first step the microcomputer divides our time series into two roughly equal parts, and computes the mean of each (8.1% and 9.1% respectively.)



In the second step the microcomputer fits a line through the two means. The points on the line represent our forecasts.

Figure 2

Percent Changes

The Percent Change routine works as its name implies. The microcomputer first computes the percent delta between the last two values of our time series: 100*(9.69 — 9.52)/9.52 =

1.79%, or 1.0179 in index form. Then it applies this factor to future periods.

Hence, the predicted interest rate in May 1984 is 9.69*1.0179 = 9.86%.

The technique of Percent Changes is usually best suited to short-term forecasts, say up to two or three periods ahead and for cases where observations are highly correlated from one period to another. Examples include using monthly data to forecast next quarter's Dow Jones Average or foreign trade deficit.

As a general rule, leave long-term prognostications to Least-Squares Trend or to the method of Semi-Averages. An exception is when you strongly believe that very recent data will heavily influence the future. An example might be a sharp jump in the inflation rate which in turn fuels the fear of future price hikes and, hence, becomes a self-fulfilling prophecy.

First Differences

The First Difference routine works a lot like its Percent Change colleague. Namely, the difference instead of percent change between the last two values of the time series is used as the factor to forecast future values.

For our data, this difference is 9.69 — 9.52 = 0.17. Hence, interest rate forecasts for May and June using this technique are 9.86% (9.69 + 0.17) and 10.03% (9.86 + 0.17), respectively.

Once again, the method of First Differences is a short-run forecasting tool. You might want to compare it to Percent Changes in making 2 or 3 period forecasts. Your predictions are probably strongest where the two methods agree and weakest where they diverge. The comparison, in other words, should give you a "warm fuzzy" or a "rigid frigid", or perhaps both.

Past Averages

Finally, the Past Average routine computes Percent Changes and First Differences using any number of past values, with you telling the microcomputer how many.

For example, the mean of the past two first differences is (9.69 — 9.52) + (9.52 — 9.09) all divided by 2, or 0.30. The microcomputer in this case predicts a May 1984 interest rate of 9.69 + 0.30 = 9.99%.

The Past Average procedure is ideally suited to forecasting situations where a string or subset of past values is deemed most important. It ignores the long-term drift captured by Least-Squares Trend and the method of Semi-Averages, and avoids the short-run bias of Percent Changes and First Differences.

Summary

A time series is a group of observations on a variable, in chronological order, at a set frequency. Your microcomputer program forecasts future values of a time series using these extrapolation techniques: (1) Least-Squares Trend, (2) Semi-Averages, (3) Percent Changes, (4) First Differences and (5) Past Averages.

All five methods suggest that interest rates will rise in the near future. And all five invoke the adage "The past is prologue." If it isn't, then perhaps we ought to have Madame Zelna read our palms!

Listing Notes:

Brian Flynn provided the original programs for the Apple II. Mike Rowe has modified these so that they can work on the Commodore 64, Atari and the CoCo (with Flex), as well as the Apple. The Listing 1 contains the "generic" code. This will not work without the system specific subroutines provided below. Listings 2 through 5 provide the required subroutines for the Apple, Commodore 64, Atari and CoCo (with Flex) respectively. These additional routines must be added to Listing 1 in order for the program to work on any microcomputer.

Listing 1

```
  1Ø REM TIME-SERIES FORECASTING
  2Ø REM BY BRIAN FLYNN
  3Ø REM MAY 1984
  32 REM MODIFIED FOR COMMODORE 64, ATARI AND FLEX
  34 REM BY MIKE ROWE, JULY 1984
  4Ø REM INITIALIZE
  5Ø GOSUB 1ØØØ
  6Ø REM ENTER DATA
  7Ø GOSUB 3ØØØ
  8Ø REM EDIT DATA
  9Ø GOSUB 35ØØ
 1ØØ REM CHOOSE METHOD
 11Ø GOSUB 5ØØØ
 12Ø REM FORECAST
 13Ø IF CH <> 6 THEN GOSUB 55ØØ: GOTO 11Ø
 14Ø END
 ***********************************************
 INSERT SUBROUTINES FOR YOUR MICROCOMPUTER HERE
 ***********************************************
1ØØØ REM INITIAIZE
1Ø1Ø REM TITLE
1Ø15 DIM C$(6)
1Ø2Ø GOSUB 15ØØ
1Ø3Ø REM INSTRUCTIONS
1Ø4Ø GOSUB 2ØØØ
1Ø5Ø REM CHOICES
1Ø6Ø GOSUB 25ØØ
1Ø7Ø RETURN
15ØØ REM TITLE
151Ø GOSUB 3ØØ
152Ø VT=1Ø: HT=15: GOSUB 4ØØ: PRINT "TIME-SERIES"
153Ø VT=11: HT=15: GOSUB 4ØØ: PRINT "FORECASTING"
154Ø FOR D = 1 TO 1ØØØ: NEXT D
155Ø RETURN
2ØØØ REM INSTRUCTIONS
2Ø1Ø REM MAXIMUM NUMBER OF OBSERVATIONS
2Ø2Ø DATA 15Ø
2Ø3Ø READ MN
2Ø4Ø DIM Y(MN)
2Ø5Ø REM INSTRUCTIONS
2Ø6Ø GOSUB 3ØØ
2Ø7Ø PRINT " THIS PROGRAM FORECASTS FUTURE VALUES"
2Ø8Ø PRINT "OF A TIME SERIES USING A HOST OF TREND-"
2Ø9Ø PRINT "ANALYSIS TECHNIQUES."
21ØØ PRINT
211Ø PRINT " THE MAXIMUM NUMBER OF OBSERVATIONS"
212Ø PRINT "ALLOWED IS ";MN;"."
213Ø PRINT
214Ø PRINT "CHANGE LINE 2Ø2Ø FOR A DIFFERENT LIMIT."
215Ø VT=22: HT=14: GOSUB 4ØØ:
     PRINT "PRESS ANY KEY ";
216Ø GOSUB 6ØØ
217Ø RETURN
25ØØ REM CHOICES
251Ø DATA  LEAST-SQUARES TREND, SEMI-AVERAGES,
     PERCENT CHANGES
252Ø DATA  FIRST DIFFERENCES, PAST AVERAGES, NONE
```

```
2530 FOR I = 1 TO 6
2540 READ C$(I)
2550 NEXT I
2560 RETURN
3000 REM ENTER DATA
3010 GOSUB 300
3020 PRINT " PLEASE ENTER OBSERVATIONS ON YOUR TIME"
3030 PRINT "SERIES. HIT 'RETURN' WHEN THROUGH."
3040 N = MN
3050 BK$ = "      "
3060 FOR I = 1 TO MN
3070 VT=6: HT=1: GOSUB 400
3075 PRINT "PERIOD #( ";I;") = ";
3080 GOSUB 700
3090 IF XX$ = "" THEN N = I-1: I = MN: GOTO 3120
3100 IF XX$ <> "" THEN Y(I) = VAL(XX$)
3110 VT=6: HT=18: GOSUB 400: PRINT BK$
3120 NEXT I
3130 REM CHECK FOR ENOUGH DATA
3140 IF N > 2 THEN RETURN
3150 VT=21:HT=1:GOSUB 400
3160 PRINT "SORRY, AT LEAST 3 OBSERVATIONS NEEDED !":
     GOSUB 800: GOTO 3040
3500 REM EDIT DATA
3510 FOR L = 0 TO INT((N-1)/10)
3520 REM DISPLAY DATA
3530 GOSUB 4000
3540 REM CORRECT DATA
3550 GOSUB 4500
3560 NEXT L
3570 RETURN
4000 REM DISPLAY DATA
4010 GOSUB 300
4020 PRINT "THESE ARE VALUES OF YOUR TIME SERIES:"
4030 FOR J = 1 TO 10
4040 M = J+L*10
4050 IF M > N THEN 4060
4055 VT=J+3: HT=1: GOSUB 400:
     PRINT "PERIOD ( ";M;" ) = ";Y(M)
4060 NEXT J
4070 RETURN
4500 REM CORRECT DATA
4510 VT=16: HT=1: GOSUB 400:
     PRINT "CORRECTIONS (Y/N) ? ";
4520 GOSUB 800: GOSUB 600
4530 IF XX$ = "N" THEN 4660
4540 IF XX$ <> "Y" THEN 4520
4550 VT=18: HT=1: GOSUB 400:
     PRINT "WHAT IS THE NUMBER OF THE DATUM TO"
4560 VT=19: HT=16: GOSUB 400: PRINT BK$: GOSUB 800
4565 VT=19: HT=1:GOSUB 400
4570 PRINT "BE CORRECTED ? ";: GOSUB 700
4590 Q = VAL(XX$)
4600 IF Q >=(1+L*10) OR Q <=N OR Q <=(10+L*10)
     THEN 4610
4605 VT=21:HT=1:GOSUB 400
4606 PRINT "OUTSIDE BOUNDS SHOWN. PLEASE TRY AGAIN.":
     GOTO 4560
4610 VT=23: HT=1: GOSUB 400: GOSUB 800
4620 PRINT "NEW VALUE = ";: GOSUB 700
4640 Y(Q) = VAL(XX$)
4650 GOSUB 4000: GOTO 4510
4660 RETURN
5000 REM CHOOSE METHOD
5010 GOSUB 300
5020 PRINT "FUTURE VALUES OF YOUR TIME SERIES ARE"
5030 PRINT "PROJECTED USING ANY OF THESE METHODS:"
5040 FOR I = 1 TO 6
5050 VT=2*I+3: HT=10: GOSUB 400: PRINT I;". ";C$(I)
5060 NEXT I
5070 VT=19: HT=10: GOSUB 400:
     PRINT "YOUR CHOICE = ? ";
5080 GOSUB 800: GOSUB 600
5090 CH = VAL(XX$)
5100 IF CH < 1 OR CH > 6 THEN 5080
5110 RETURN
5500 REM MAKE PROJECTIONS
5510 REM NUMBER OF FUTURE PERIODS
5520 GOSUB 6000
5530 REM PROJECTIONS
5540 GOSUB 300
5550 IF CH <> 5 THEN VT=12: HT=13: GOSUB 400:
     PRINT "FORECASTING ...";
5560 ON CH GOSUB 6500,7000,7500,8000,8500
5570 REM DISPLAY
5580 GOSUB 11500
5590 RETURN
6000 REM NUMBER OF FUTURE PERIODS
6010 GOSUB 300
6020 PRINT "THE LAST PERIOD OF YOUR TIME SERIES IS"
6030 PRINT "NUMBER ";N;"."
6040 PRINT
6050 PRINT "HOW MANY PERIODS INTO THE FUTURE DO YOU"
6060 VT=5: HT=20: GOSUB 400: PRINT BK$;: GOSUB 800:
     VT=5: HT=1: GOSUB 400
6070 PRINT "WANT TO FORECAST ? ";: GOSUB 700
6080 NF = VAL(XX$)
6090 IF NF < 1 THEN 6060
6100 REM CHECK FOR ENOUGH MEMORY
6110 T = N+NF: IF T <= MN THEN RETURN
6120 VT=22: HT=1: GOSUB 400
6125 PRINT "SORRY, ONLY ";MN-N;
     " MORE PERIODS ALLOWED.";: GOTO 6060
6130 RETURN
6500 REM LEAST-SQUARES TREND
6510 REM KEY SUMS
6520 SX = 0:SY = 0:XQ = 0:YQ = 0:CP = 0
6530 FOR I = 1 TO N
6540 SX = SX+I
6550 SY = SY+Y(I)
6560 XQ = XQ+I*I
6570 YQ = YQ+Y(I) ↑ 2
6580 CP = CP+Y(I)*I
6590 NEXT I
6600 REM A & B
6610 B = (N*CP-SX*SY)/(N*XQ-SX*SX)
6620 A = (SY-B*SX)/N
6630 REM FORECASTS
6640 FOR I = N+1 TO T
6650 Y(I) = A+B*I
6660 NEXT I
6670 RETURN
7000 REM METHOD OF SEMI-AVERAGES
7010 REM NUMBER OF POINTS IN EACH GROUP
7020 G1 = INT(N/2)
7030 G2 = N-G1
7040 REM GROUP MEANS
7050 Y1 = 0:X1 = 0
7060 FOR I = 1 TO G1
7070 Y1 = Y1+Y(I)
7080 X1 = X1+I
7090 NEXT I
7100 Y1 = Y1/G1:X1 = X1/G1
7110 REM MEANS OF SECOND GROUP
7120 Y2 = 0:X2 = 0
7130 FOR I = G1+1 TO N
```

```
7140 Y2 = Y2+Y(I)
7150 X2 = X2+I
7160 NEXT I
7170 Y2 = Y2/G2:X2 = X2/G2
7180 B = (Y2-Y1)/(X2-X1)
7190 REM Y-INTERCEPT
7200 A = Y2-B*X2
7210 REM FORECASTS
7220 FOR I = N+1 TO T
7230 Y(I) = A+B*I
7240 NEXT I
7250 RETURN
7500 REM PERCENT CHANGE
7510 FOR I = N+1 TO T
7520 Y(I) = Y(I-1)*Y(I-1)/Y(I-2)
7530 NEXT I
7540 RETURN
8000 REM FIRST DIFFERENCE
8010 FOR I = N+1 TO T
8020 Y(I) = 2*Y(I-1)-Y(I-2)
8030 NEXT I
8040 RETURN
8500 REM PAST AVERAGE
8510 PRINT "WOULD YOU LIKE TO USE AN AVERAGE OF"
8520 PRINT "PAST:"
8530 VT=4: HT=10: GOSUB 400: PRINT "1. % CHANGES"
8540 VT=6: HT=10: GOSUB 400:
     PRINT "2. FIRST DIFFERENCES"
8550 VT=8: HT=10: GOSUB 400:
     PRINT "3. ACTUAL VALUES"
8560 VT=12: HT=10: GOSUB 400: PRINT "CHOICE = ? ";
8570 GOSUB 800: GOSUB 600
8580 AV = VAL(XX$)
8590 IF AV < 1 OR AV > 3 THEN 8570
8600 IF AV = 1 THEN T$ = "PERCENT CHANGES"
8610 IF AV = 2 THEN T$ = "FIRST DIFFERENCES"
8620 ON AV GOSUB 9000,9000,9500
8630 GOSUB 300
8640 VT=12: HT=13: GOSUB 400:
     PRINT "FORECASTING ..."
8650 ON AV GOSUB 10000,10500,11000
8660 RETURN
9000 REM % CHANGES OR FIRST DIFFERENCES
9010 VT=15: HT=1: GOSUB 400:
     PRINT "HOW MANY PAST ";T$
9020 VT=16: HT=22: GOSUB 400: PRINT BK$: GOSUB 800
9030 VT=16: HT=1: PRINT "DO YOU WANT TO USE ? ";:
     GOSUB 700
9040 PP = VAL(XX$)
9050 IF PP < 1 THEN 9020
9060 IF PP < N THEN RETURN
9070 VT=22: HT=1: GOSUB 400
9080 PRINT "SORRY, ONLY ";N-1;" ARE AVAILABLE":
     GOTO 9020
9500 REM ACTUAL VALUES
9510 VT=15: HT=1: GOSUB 400:
     PRINT "HOW MANY PAST ACTUAL VALUES WOULD"
9520 VT=16: HT=19: GOSUB 400: PRINT BK$: GOSUB 800
9530 VT=16: HT=1: GOSUB 400:
     PRINT "YOU LIKE TO USE ? ";: GOSUB 700
9540 PP = VAL(XX$)
9550 IF PP < 1 THEN 9520
9560 IF PP <= N THEN RETURN
9570 VT=22: HT=1: GOSUB 400
9580 PRINT "SORRY, ONLY ";N;" ARE AVAILABLE":
     GOTO 9520
10000 REM PERCENT CHANGES
10010 REM PAST AVERAGE
10020 PC = 0
10030 FOR I = N TO N-PP+1 STEP-1
10040 PC = PC+Y(I)/Y(I-1)
10050 NEXT I
10060 PC = PC/PP
10070 REM FORECASTS
10080 FOR I = N+1 TO T
10090 Y(I) = Y(I-1)*PC
10100 NEXT I
10110 RETURN
10500 REM FIRST DIFFERENCES
10510 REM PAST AVERAGE
10520 FD = 0
10530 FOR I = N TO N-PP+1 STEP-1
10540 FD = FD+Y(I)-Y(I-1)
10550 NEXT I
10560 FD = FD/PP
10570 REM FORECASTS
10580 FOR I = N+1 TO T
10590 Y(I) = Y(I-1)+FD
10600 NEXT I
10610 RETURN
11000 REM ACTUAL VALUES
11010 REM PAST AVERAGE
11020 AC = 0
11030 FOR I = N TO N-PP+1 STEP-1
11040 AC = AC+Y(I)
11050 NEXT I
11060 AC = AC/PP
11070 REM FORECASTS
11080 FOR I = N+1 TO T
11090 Y(I) = AC
11100 NEXT I
11110 RETURN
11500 REM DISPLAY
11510 FOR L = 0 TO INT((NF-1)/10)
11520 REM HEADING
11530 GOSUB 12000
11540 REM BODY
11550 GOSUB 13000
11560 NEXT L
11570 RETURN
12000 REM HEADING
12010 GOSUB 300
12020 F$ = "======================================="
12030 PRINT F$
12040 VT=2: HT=16: GOSUB 400: PRINT "FORECASTS"
12050 PRINT F$
12060 VT=5: HT=1: GOSUB 400: PRINT "METHOD: ";C$(CH);
12070 IF CH = 5 THEN GOSUB 12500
12080 RETURN
12500 REM PAST AVERAGE
12510 PRINT " OF ";PP
12520 VT=6: HT=9: GOSUB 400
12530 IF AV = 1 THEN PRINT "% CHANGES"
12540 IF AV = 2 THEN PRINT "FIRST DIFFERENCES"
12550 IF AV = 3 THEN PRINT "ACTUAL VALUES"
12560 RETURN
13000 REM BODY
13010 FOR J = 1 TO 10
13020 M = J+L*10+N
13030 IF M > T THEN 13040
13035 VT=J+7: HT=1: GOSUB 400:
      PRINT "PERIOD ( ";M;" )= ";Y(M)
13040 NEXT J
13050 VT=21: HT=1: GOSUB 400: PRINT F$
13060 VT=22: HT=14: GOSUB 400:
      PRINT "PRESS ANY KEY ";
13070 GOSUB 600
13080 RETURN
```

Subroutines

```
200  REM   FLEX SUBROUTINES

299  REM   ** CLEAR DISPLAY **
300  PRINT CHR$(11);CHR$(27);"X";CHR$(24);:RETURN

399  REM   ** POSITION CURSOR **
400  IF VT> Ø THEN PRINT CHR$(11);:
     FOR II=1 TO VT:PRINT:NEXT II
410  IF HT> Ø THEN PRINT TAB(HT);
420  RETURN

499  REM   ** POSITION CURSOR AND SPACE **
500  GOSUB 400: PRINT SPC(SP);: RETURN

599  REM   ** GET CHARACTER ROUTINE **
600  INPUT XX$: IF XX$="X" THEN XX$=""
610  RETURN

699  REM   ** INPUT ROUTINE **
700  GOTO 600

799  REM  ** MAKE SOUND (OPTIONAL) **
800  RETURN : REM ADD CODE HERE TO MAKE A
801  REM  SOUND IF YOU SO DESIRE !!
```

```
*********************************

200 REM   COMMODORE SUBROUTINES

299 REM   ** HOME AND CLEAR DISPLAY **
300 PRINT "{CLEAR}";:RETURN

399 REM   ** POSITION CURSOR **
400 PRINT "{HOME}";
410 FOR XX=1 TO VT:PRINT :NEXT XX
420 IF HT> Ø THEN PRINT TAB(HT);
430 RETURN

499 REM  ** POSITION CURSOR AND SPACE **
500 GOSUB 400: PRINT SPC(SP);: RETURN

599 REM  ** GET SUBROUTINE **
600 XX$=""
610 GET XX$: IF XX$="" THEN 610
620 RETURN

699 REM  ** INPUT SUBROUTINE **
700 PRINT {SPACE10,BACKSPACE10};: INPUT XX$: RETURN

799 REM  ** MAKE SOUND (OPTIONAL) **
800 RETURN : REM  ADD CODE TO MAKE A
801 REM  SOUND IF YOU SO DESIRE !!!

*********************************
```

```
*********************************

200  REM   APPLE II SUBROUTINES

299  REM  ** HOME AND CLEAR DISPLAY **
300  HOME : RETURN

399  REM  ** POSITION CURSOR **
400  IF VT> Ø THEN VTAB(VT)
410  IF HT> Ø THEN HTAB(HT)
420  RETURN

499  REM  ** POSITION CURSOR AND PRINT SPACES **
500  GOSUB 400: PRINT SPC(SP);: RETURN

599  REM  ** GET SUBROUTINE **
600  GET XX$: RETURN

699  REM  ** INPUT SUBROUTINE **
700  INPUT XX$: RETURN

799  REM  ** MAKE SOUND **
800  PRINT CHR$(7);: RETURN

*********************************
```

```
*** ATARI VERSION ***

200 REM ATARI SUBROUTINES

299 REM ** HOME AND CLEAR DISPLAY **
300 PRINT CHR$(125);:RETURN

399 REM ** POSITION CURSOR **
400 POSITION HT,VT:RETURN

499 REM ** POSITION CURSOR AND PRINT SPACES **
500 GOSUB 400
510 FOR I=1 TO SP:PRINT CHR$(32):NEXT I:RETURN

599 REM ** GET SUBROUTINE **
600 GET #1,X:XX$=CHR$(X):RETURN

699 REM ** INPUT SUBROUTINE **
700 INPUT XX$:RETURN

799 REM ** MAKE SOUND  **
800 RETURN
810 REM ADD YOU OWN SOUND IF DESIRED

*** THE NEXT TWO LINES MUST BE ADDED ***

1015 DIM BK$(10),XX$(10),T$(20),F$(40)
1016 OPEN #1,4,0,"K:"

*** LINE 1060 AND LINES 2500 TO 2560 MUST BE DELETED.

*** REPLACE LINES 5050 AND 12060 AS FOLLOWS:

5050 VT=2*I+3:HT=10:GOSUB 400:PRINT I;".  ";:
     GOSUB 14000

12060 VT=5:HT=1:GOSUB 400:PRINT "METHOD: ";:
     GOSUB 14005

*** ADD THE FOLLOWING SUBROUTINE

14000 ON I GOTO 14010,14020,14030,14040,14050,14060
14005 ON CH GOTO 14010,14020,14030,14040,14050,14060
14010 PRINT "LEAST-SQUARES TREND";:RETURN
14020 PRINT "SEMI-AVERAGES";:RETURN
14030 PRINT "PERCENT CHANGES";:RETURN
14040 PRINT "FIRST DIFFERENCES";:RETURN
14050 PRINT "PAST AVERAGES";:RETURN
14060 PRINT "NONE";:RETURN
```

books

Title: Mastering Your Atari Through Eight BASIC Projects
Author: the staff of MICRO magazine, Tom Marshall ed.
Price: $19.95, disk included
Publisher: Prentice-Hall

Using a 'learning-by-doing' approach the reader is quickly taught how to write, modify, and expand his own programs.
A diskette is included which contains complete running programs to begin with. The eight projects include Micro Calc, a miniature spreadsheet; Master, a guessing game; Atari Clock; Word Detective; Atari Player, a music program; Breakup, an exciting game; Sorting, sorts graphic bars and telephone directory; Programmable Characters, add extra plotting resolution while retaining most normal characters in the Atari character set. Each project is designed to teach the reader a specific aspect of programming — string manipulation, BASIC functions, random numbers and flags, ON...GOSUB, character graphics, animation, sorting methods, plus many more. There are listings for all the programs, clear operating instructions, examples and figures. Each programming element is explained with clarity and related to the project the reader is working on. This book presents learning and mastery with hands-on experience and fun. This is the second in a series of 'Mastering' books, the first was for the Vic-20, the next is for the Commodore 64.

Level: Beginner to intermediate.

Title: Getting On-Line
Author: M. David Stone
Price: $14.95
Publisher: Prentice-Hall, Inc.

The field of telecommunications is constantly growing and at a startling rate. Finding out how, what and where is a time consuming and often confusing task. It is the purpose of this book to help both the novice and old-hand at sorting out what they need, where to get it, and how to use it. The first six chapters explain what you need before you actually go on-line. Chapter 1 Information Utilities — a look at what is available and where it came from, examines kinds of information utilities and data bases. Chapter 2 Hardware Utilities — covers dumb terminals, smart terminals and computers as terminals. Chapter 3 Hardware II — deals with modems, baud rate, signaling standards, connecting a modem, direct connect vs. acoustical connect modems and choosing a modem. Chapter 4 Software Part I — a look at available features, computers as dumb and smart terminals, smart terminal programs; basic features and additional capabilities. Chapter 5 Software Part II — the nitty gritty, RS-232, short reviews and helpful hints regarding some popular micros, CP/M computers, dual processor machines and modems. Chapter 7 Search Strategy deals with how to go about organizing your search for information, with tips that can save you money and time. The rest of the book consists of a catalog of information and an index. The catalog is of the various utilities (Dow Jones, The Source, etc.), and free services (Public Access Bulletin Boards).

Level: Beginner to advanced.

Title: The Apple IIc Book
Author: Bill O'Brien
Price: $12.95
Publisher: Bantam Books

Bantam was selected by Apple to be among those who introduced the IIc. Providing information that is useful to novices and advanced users, it includes data not found in the owner's manual. Compatability, configuration, DOS 3.3 and ProDOS are all covered. The addition of peripherals features mice, touchpads, graphics tablets, and the new 'flat screen.' BASIC (Applesoft) is explained in as much as one can in three chapters. Graphics is also touched upon. Telecommunications - i.e. the connections of modems, bulletin boards, etc. are dealt with in one chapter. Another chapter is devoted to troubleshooting, hardware and software problems. The four most used applications — Word Processing, Databases, Spreadsheets, and Communications are introduced and reviewed. The last chapter contains vital information on user groups, bulletin boards, magazines and books. The appendices contain hardware information, technical stats and Escape codes. This books covers a lot of ground and hence is limited in the depth of coverage. However it does contain a great deal of useful information and has gathered some information not easily found elsewhere.

Level: beginner to advanced.

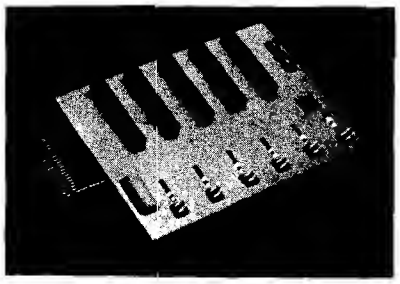
Title: Introduction to C
Author: Paul M. Chirlian
Price: $15.95
Publisher: Matrix Publishers, Inc.

As the title says this is an introduction to the programming language C. It is designed so that even those who have no previous programming experience will be able to learn C. Starting out with some basic ideas about computer operation, it moves on to some of the fundamental concepts of programming in C. Fundamental arithmetic operations are explained: integer, floating-point, hierarchy, mixed mode, constants, etc. The next area is basic input, output and character operations including the use of the 'printf' statement and strings. The author stresses structured programming and documentation, devoting a large section of the book to these topics. The debugging process is outlined, a subject seldom covered. There are a number of good exercises and over 70 example programs. Arrays, pointers, and manipulations are covered in detail. File handling is discussed, input/output redirection, disk files, and command line input of data. The appendices contain C Keywords, C Operators, and the ASCII Codes. Chirlian has chosen to follow the standards set forth by Brian W. Kernighan and Dennis M. Ritchie in their book 'The C Programming Language,' Prentice-Hall, Englewood Cliffs, N.J. 1978. This book and Chirlian's are considered the standard books on C programming.

Level: Beginner to intermediate.

MICRO

Name: **Cardboard/5**
System: Commodore 64



Description: This product allows greater flexibility of use to switch select any cartridge slot or combination of cartridge slots. The 22 color coded lights emit diodes to give status indication. Each slot has four LEDS and two toggle switches for indication and control. It allows the user supply power to a cartridge without allowing it to auto-start or to effect other operations.

Price: $79.95
Available: Cardco, Inc.
313 Mathewson
Wichita, KS 67214

Name: **Computereyes**
System: Apple II Series & Compatibles
Memory: 48K
Language: Applesoft & DOS 3.3

Description: This is a slow-scan device that connects any standard video source (video tape recorder, video camera, videodisk, etc.) and the Apple's game I/O socket. A multi-scan mode provides realistic grey-scale images. Included in the package: interface module, cable, software support on disk, owner's manual and also comes with a one year warranty. Versions for other popular computers will be available soon.

Price: $129.95
$10.00 demo disk (not required)
Available: Digital Vision, Inc.
14 Oak Street - Suite 2
Needham, MA 02192
(617) 444-9040

BOUNTY HUNTER

Journey back with us into the days of Jessie James and Billy the Kid where the only form of justice was a loaded revolver and a hangman's noose. In this full-length text adventure, you play the role of Bounty Hunter, battling against ruthless outlaws, hostile Indians, wild animals and the elements of the wilderness with only your wits and your six gun. Average solving time: 20-30 hours. If you love adventures, this one is a real treat. Available for COMMODORE 64, the VIC-20 (with expander), and COLECO ADAM. See your dealer.

$19.95 Cassette

DEALERS INVITED!

Published by:
Star-Byte, Inc.
A Division of Robinson-Halpern Company
2564 Industry Lane • Norristown, PA 19403 • 215-539-4300

Victory Software

ADAM is a trademark of Coleco, Inc. COMMODORE 64 is a trademark of Commodore Business Machines, Inc. VIC-20 is a trademark of Commodore Business Machines, Inc.

Name: **Microsport Microcomputers Model MMC/02**

System: AIM 65, Apple II, Atari 400,800, Commodore CBM/PET series, KIM-1, KIM-4, Vic-20, MTU 1300 and motherboards, Ohio Scientific 600 and others, Synertek SYM-1.

Description: The Model MMC/02 is a complete microcomputer on a 4.5" by 6.5" PC board. It features a 6502 microprocessor, 1K Ram standard, 4K ROM/EPROM socket, 2K RAM or ROM/EPROM expansion and BUS for ading up to 16 I/O devices. Three basic versions are available. The MMC can operate from a regulated plus 5VDC power source. The CPU addresses a total of 8K, enough for most control applications. There is a prototyping area as well as spare gates.

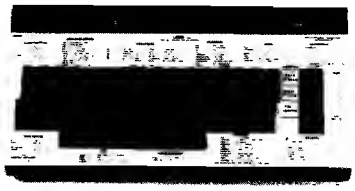
Price: From $159.00*

Available: R.J. Brachman Associates, Inc
P.O. Box 1077
Havertown, PA 19083
(215) 622-5495

Name: **Cheatsheet**

System: Commodore VIC 20, C-64

Description: These are plastic laminated keyboard overlays designed to fit over the keyboard surrounding the keys with commands and controls grouped together for easy references. The latest cheatsheets available are: Logo (sheet 1), Logo (sheet 2, advanced), Pilot, Easy Calc, Printer-1526, The Manager, Multiplan, Practicalc 64 (& plus), Printer (Epson-RX-80), Superbase 64, The Consultant, Sprites Only and blanks. This brings the total cheatsheets available to 33.



Price: $3.95 plus $1.00 shipping per order

Available: Cheatsheet Products
P.O. Box 8299
Pittsburgh, PA 15218
(412) 456-7420

EMC

UNLIMITED PROFIT POTENTIAL

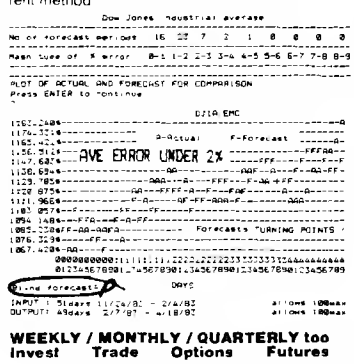
WATCH OUT WALL STREET!! Now available on floppy disk for brokers and active traders. The AMAZING computer program **«STOCKER1»** uses new Moving Window-Spectral algorithm to forecast stock/commodity market **TURNING POINTS**—not mere trend line/moving averages. Affordable! Easy to use!

USED BY STOCK/COMMODITY BROKERS & ACTIVE TRADERS

Perhaps you too are ready for the STOCKER1 challenge !

N E W ! *Your own personal forecast.*

COMPARE *There is just nothing quite like it* **« STOCKER1 »** forecast for the Dow Jones industrial average daily closing with your current method



« STOCKER1 » available for

IBM, PC, PC Jr, XT *MSDOS*
APPLE II, II+, IIe-
with Z-80 card
TRS80 MODII I, 4,
II, 12, 16 4/5 diskette

$299 MASTER/VISA/CHECK
INCLUDES FREE HOUR ON MODEM SERVICE

ENGINEERING MANAGEMENT CONSULTANT

P.O. Box 312

Fairfax, Va. 22030

Tel.: (703) 425-1296.

WEEKLY / MONTHLY / QUARTERLY too
Invest Trade Options Futures

Does your broker use STOCKER1 ?

microbe

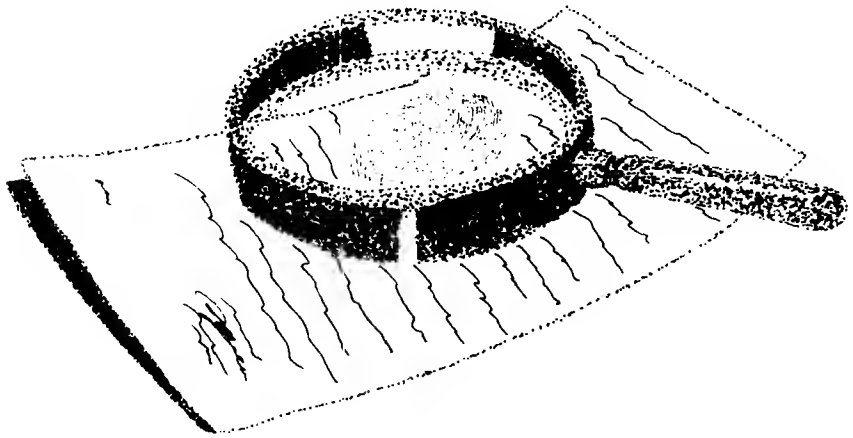
In Micro No. 73 (July) Ian R. Humphreys 'CMPRSS' program the following lines should read:

```
9002 8D F6 03          STA BJP+1
9007 8D F7 03          STA BJP+2
92E5 A9 A9             LDA #< MESS1
932E A9 EE     PRT1A   LDA #< MESS1A
9400 69 FF             ADC #$FF
9402 85 03             STA NEWPTR+1
```

NOTE: It is the policy of Micro to not include all of the hex code for assembled text. Due to space limitations we include only the first three bytes of the assembled message. This practice is carried throughout all of our assembler listings. We apologize for any confusion this may have presented and also any inconvenience to those who are not using an assembler.

lyte bytes

by Mark S. Morano



It all started two months ago when I received a call from the BES (Bureau of Encryption Services). They were having trouble with data security between their micros and mainframes. The problem was thought to be originating from inside. Regardless of what new encryption methods were developed, the unknown informant would render them useless. And so the Bureau had decided to seek outside help. Due to our years of experience in the field, Micro was given the job. Our resident expert in this area, Mike Rowe, was assigned the task of developing new encryption methods that would prove effective against prying thieves.

All went well until three weeks ago when Mike went on vacation. He called on the day he was to return saying he'd been delayed by a death in the family. We immediately informed the BES of the delay and the reason for it. The following day the Bureau contacted his family. They hadn't heard from him. Needless to say there had been no funeral. We were told to call the Bureau immediately if we heard from Mike. They assured us that they would locate him, but a week passed and still no sign of Mike. I began to worry. I have known Mike for a long time and knew he wasn't the kind of guy to up and disappear. The whole thing sounded strange. That is, until the other night.

I was working late, downloading some files from a local mainframe we use for mass storage. While looking at the catalog I noticed a file named 'test.mr'. It was nothing unusual to find a test file, but we never used the extension 'mr'. I downloaded the file to check it out. At first it seemed to be a garbled mess, as though something strange happened during transmission. I decided that there must have been a surge on the wires, or a lost handshake somewhere, so I downloaded again. As I did I thought about the strange extension — 'mr' — of course, Mike Rowe — it must be one of his work files, but he always named his 'temp.tst'. I started to examine the file more closely; it still seemed like a bad transmission to me. Perhaps I would find a clue on his desk. After rummaging around piles of paper, I found what I was looking for, a folder marked 'Top Secret' in big red letters with hand-painted stars and spaceships scattered about. With folder in hand I went back to my desk. I printed out Mike's file, carefully perused it, and then started pouring through the 'Top Secret' folder in hope of an answer. As I sifted through countless encryption methods, I suddenly came upon one dated three days before Mike went on vacation. He had mentioned he was onto something hot and had pulled an all nighter the Thursday before he left. As I compared his examples and 'test.mr' I knew I had it. Two hours later I had decoded the mystery file.

At first I thought it was a gag, but it soon became evident it wasn't. The text explained that he had been picked up by a couple of woman; the next thing he remembered was passing out. He woke to find himself locked in a room with a desk, a couple of terminals, a printer and a modem. He was instructed over an intercom to recompose the latest encryption method he had devised. He was able to convince his captors that he had to access some files from the mainframe Micro used. While online he sent over this dummy work file I had found. I later found out he had given this new encryption method to the Bureau before he left for vacation. It seems that now someone else wanted it too. Mike stated in 'test.mr' that his method would be used to send important data over the lines sometime during the next month. He asked me to inform the Bureau of his plight, destroy 'test.mr' and any files associated with his work. His last request was that I quickly come up with some alternate encryption methods for the BES to use in place of his.

Well, in all honestly, I have limited experience in this area. So I decided to tap the brains of some people I know. After considering different suggestions, I finally came to the conclusion that you, the readers of Micro, could probably help the most. By gathering a variety of encryption methods, I could distill one final product that would ensure security and stymy any intruder. And so, I ask you to help me out in this time of need. Please send your encryption schemes and solutions to me, using the sentence 'When the crow flies west, the sun shall set in the east.' Any encryption methods you can or have devised will be of great value and an important link in forging a chain that only Mike Rowe himself could crack.

Advertiser's Index

Analog	6
Call A.P.P.L.E.	1
Cardco, Inc.	69
Cheatsheet Products	70
Digital Vision	69
Engineering Management	70
Hayes Microcomputer Products	Back Cvr
JQB Enterprises	46
Lazerware	35
MICRO	60,Ins Back Cvr
Micro Motion	8,30
Micro Technology Unlimited	3
Micro-W Distributing	17
Midwest Micro	9
Nikrom Technical Products	2
Performance Micro	17
Progressive Peripherals	12
Protecto	38,39,40,41,42,43
Quantum Software	45
R.J. Brachman	70
S-C Software	45
Skyles Electric Works	Ins Front Cvr
Specialty Electronics	14
Star Byte	69
Such-A-Deal Software	53

Coming in October —

- ☐ Plotting Binary Trees
  by Luther K. Branting
  Graphic displays of tree-like decision paths aid using and understanding this type of graph

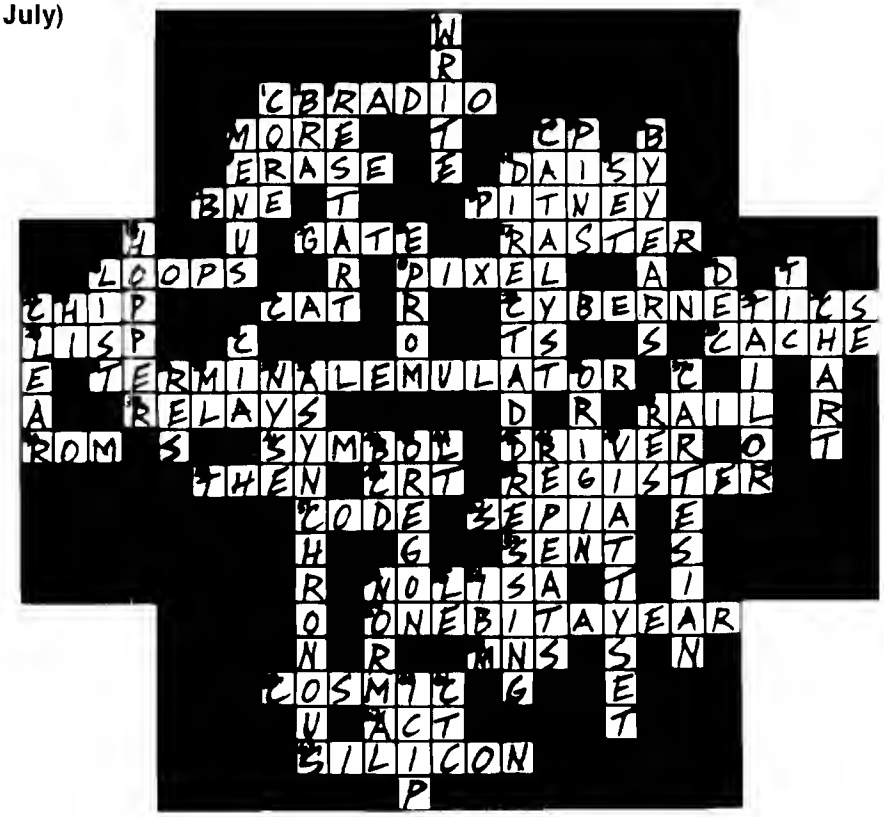
- ☐ Fat Bit Map Plotting
  by Loren Wright
  Assembly language routines to support bit map plotting on the Commodore 64

- ☐ Data Base Comparisons
  by Sanjiva Nath
  Discussion of the features to look for in selecting a data base manager for your micro

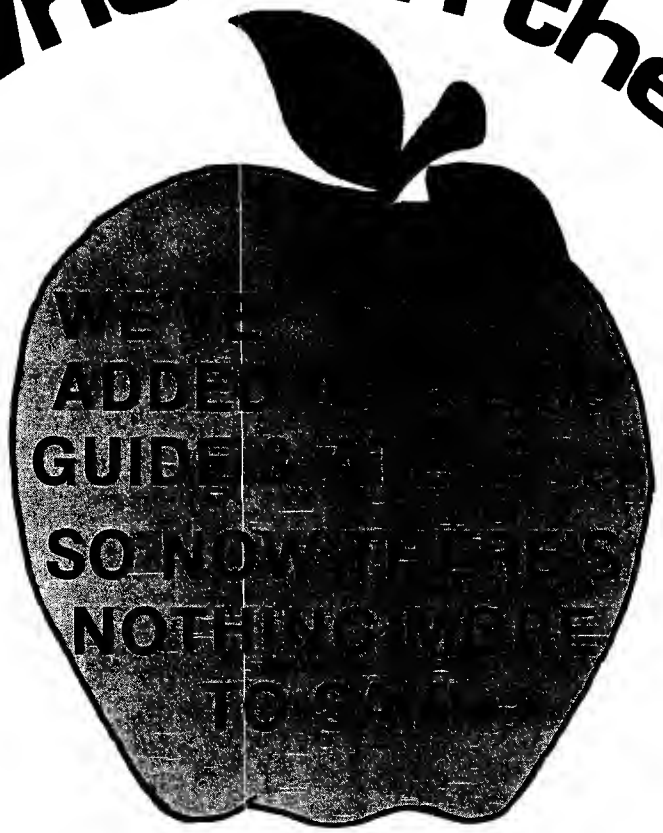
- ☐ Rational Joystick Interfacing
  by Charles Engelsher
  A 'built-it yourself' project to add a joystick to your system and learn about A/D conversion

- ☐ FORTH Input Utility
  by Mike Dougherty
  A method of providing interactive text input for better applications

Solution to last month's (#74 July) Lyte Bytes puzzle.



What's Where in the APPLE



This famous book now contains the most comprehensive description of firmware and hardware ever published for *the whole Apple II family*. A new section with guide, atlas and gazeteer now provides Apple IIe specific information.

- Gives names and locations of various **Monitor, DOS, Integer BASIC** and **Applesoft** routines and tells what they're used for
- Lists **Peeks, Pokes** and **Calls** in over 2000 memory locations
- Allows easy movement between BASIC and Machine Language
- Explains how to use the information for easier, better, faster software writing

This expanded edition is available at the new low price of only $19.95

For the 35,000 people who already own previous editions, the IIe Appendix is available separately for just $5.00.

Please send me:

_____ What's Where in the Apple @ $19.95 ea. _____
(Plus $2.00 per copy shipping/handling)

_____ Apple //e Appendix @ $5.00 ea. _____
(includes shipping charges)

Mass residents add 5% sales tax $ _____

Total Enclosed $ _____

Name _____

Address _____

City _____ State _____ Zip _____

Signature _____

☐ Check ☐ VISA ☐ MasterCard

Acct # _____ Expires _____

For faster service
Phone 617/256-3649

MICRO, P.O. Box 6502, Chelmsford, MA 01824

"My Apple's telephone just called up the home office!"

Communicating is so easy with a complete telecomputing system from Hayes. Just plug it in—and the world is your Apple. Hayes Smartmodem 300™ is the convenient direct-connect modem for the Apple IIc. And Hayes Micromodem IIe® is the easily installed board modem for the Apple II, IIe, III, and Apple Plus. Packaged with Smartcom I™ companion software, both provide a complete telecomputing system. And best of all, both systems are from Hayes, the established telecomputing leader!

Hayes, you really know how to help.

We connect you to all the right places. Bulletin boards, databases, information services—naturally. And that's just the beginning. Let your Apple plan your travel itinerary, including flight numbers, hotel and rental car reservations. Watch it retrieve and analyze daily stock and options prices. Work at home and send reports to and from your office. You can even do your gift shopping by computer!

Would you care to see our menu? Make your selection. Really. With Smartcom I, you just order up what you want to do. The program guides you along the way. You can create, list, name, send, receive, print or erase files right from the menu. From the very first time you use it, you'll find telecomputing with Hayes as easy as apple pie!

We've got your number! We know that you want a system that's flexible, versatile and accommodating. The Smartmodem 300/Smartcom I system accepts ProDOS,™ DOS 3.3, Pascal and CP/M® operating systems. The Micromodem IIe/Smartcom I system accepts DOS 3.3, Pascal and CP/M operating systems.

Smartcom I also provides you with a directory of the files stored on your disk. And will answer calls to your system, without your even being there.

Your Apple's telephone goes anywhere the phone lines go. Hayes modems allow your Apple to communicate with any Bell-103 type modem over ordinary telephone lines. You simply connect directly into a modular phone jack, to perform both Touch-Tone® and pulse dialing. Hayes Smartmodem 300 and Micromodem IIe both transmit at 110 or 300 bits per second, in either half or full duplex.

Follow the leader. Over the years we've built our reputation as the telecomputing leader by developing quality products that set industry standards. Now we invite you to see for yourself just how simple it is to add powerful, easy to use telecomputing capabilities to your Apple computer with a complete, ready-to-go system from Hayes. Visit your Hayes dealer for a hands-on demonstration. And get on line with the world.

Hayes. We're here to help.

Hayes

Hayes Microcomputer Products, Inc.
5923 Peachtree Industrial Blvd.
Norcross, Georgia 30092. 404/441-1617.

Smartmodem 300 and Smartcom I are trademarks and Micromodem IIe is a registered trademark of Hayes Microcomputer Products, Inc. Apple is a registered trademark, and ProDOS is a trademark of Apple Computer, Inc. Touch-Tone is a registered service mark of American Telephone and Telegraph. CP/M is a registered trademark of Digital Research, Inc. ©1984 Hayes Microcomputer Products, Inc.